受領書

平成１７年　２月　４日
特　許　庁　長　官

識別番号　１００１０９２１０
氏名（名称）　新居　広守　様
提出日　平成１７年　２月　４日

以下の書類を受領しました。

項番	書類名	整理番号	受付番号	出願番号通知（事件の表示）
1	国際出願	P37271-P0	50500197598	PCT/JP2005/ 1670

以　上

明 細 書

プログラム変換装置およびプログラム変換方法

技術分野

[0001] 本発明はプログラム変換装置に関し、特に実行時に外部からの所定の応答を待つ命令を含む命令セットを備えたプロセッサ向けのプログラム変換装置に関する。

背景技術

[0002] 近年、プロセッサの処理速度は急激に向上しているが、それに比べてメインメモリのアクセス速度向上は小さく、両者の速度差は年々大きくなっている。このため、情報処理装置の高速処理においてメモリアクセスがボトルネックとなることが従来指摘されている。

[0003] この問題を解消するために、記憶階層の考え方からキャッシュ機構が用いられている。キャッシュ機構では、プロセッサで必要とされるデータを主記憶から高速なキャッシュへ予め転送(プリフェッチ)しておく。これにより、プロセッサからのメモリアクセスに高速に対応することが可能である。

[0004] しかし、プロセッサがキャッシュ上に無いデータにアクセスした場合にはキャッシュミスが発生してしまう。このため、主記憶からキャッシュへのデータの転送時間がかかってしまうという問題がある。

[0005] ユーザがキャッシュを意識することなくプログラミングを行い、そのプログラムが実行されれば、このようなキャッシュミスが頻発することが想定される。その結果、キャッシュミスによるペナルティがプロセッサの性能を大きく劣化させることになる。そのため、コンパイラがキャッシュを考慮した最適化を行う必要がある。

[0006] キャッシュ最適化の技術の一つとしてプリフェッチ命令の挿入が挙げられる。プリフェッチ命令とは、あるメモリアドレスの参照が起こる前に、そのアドレスのデータを前もって主記憶からキャッシュへ転送しておくものである。プリフェッチ命令の挿入による最適化では、当該メモリアドレスの参照が起こる少し前のサイクルにプリフェッチ命令の挿入をおこなうものである。

[0007] 例えば、図1(a)に示すようなループ処理に対しては、図1(b)に示すようにデータ

が参照されるまでのレイテンシを考慮し、数イタレーション先で参照されるデータをプリフェッチするようにプリフェッチ命令(dpref())がループ内に挿入される。なお、ここでは、int型の配列aの要素は4バイト、キャッシュのラインサイズを128バイトとする。

発明の開示

発明が解決しようとする課題

[0008] しかし、図1(b)に示すコードでは、1イタレーションにつき配列aの参照とプリフェッチとがそれぞれ行われているが、参照は4byteずつしか行われないのに対して、プリフェッチは1ライン(128byte)単位で行われる。

[0009] よって、1回のプリフェッチで32回分の参照に対応できるため、残り31回は無駄にプリフェッチが行われていることになる。すなわち、同じラインのプリフェッチ命令を連続発行してしまっている。

[0010] また、プロセッサによっては、dpref命令によるデータ転送中に、次のdpref命令を実行しようとすると、前のdpref命令による主記憶からキャッシュへのデータ転送が終了していないにもかかわらず、次のdpref命令が発行されてしまい、本来インターロックを解消するためにdpref命令を挿入したにも関わらず、インターロックが起こってしまう。

[0011] よって、上記のようにループの1イタレーションが短く、2つのdpref命令の間隔が短いと、dpref命令による主記憶からキャッシュへのデータ転送にかかる時間(レイテンシ)が顕在化し、かえって性能を悪化させてしまう。

[0012] また、dpref命令の実行時以外であっても、メモリアクセス命令などのように、命令発行後に何らかの応答待ちが発生するような命令の場合であっても、インターロックを起こす可能性がある。

[0013] 本発明は、上述の課題を解決するためになされたもので、インターロックを引き起こす可能性のある命令を無駄に発行せずに、プログラム実行時の処理速度を向上させるプログラム変換装置およびプログラム変換方法を提供することを目的とする。

[0014] また、命令発行後に何らかの応答待ちが発生する命令を無駄に発行せずに、プログラム実行時の処理速度を向上させるプログラム変換装置およびプログラム変換方法を提供することを目的とする。

[0015] また、プログラム実行時にインターロックを引き起こさないプログラム変換装置およびプログラム変換方法を提供することを目的とする。

課題を解決するための手段

[0016] 上記目的を達成するために、本発明に係るプログラム変換装置は、実行時に外部からの所定の応答を待つ命令を含む命令セットを備えたプロセッサ向けのプログラム変換装置であって、入力プログラムに含まれる繰返し回数がx回のループを、繰返し回数がy回のループを内側ループとし、繰返し回数がx／y回のループを外側ループとするネスト構造への変換である二重ループ変換を行うループ構造変換手段と、前記内側ループの外部の位置に、前記命令を配置することにより、当該命令を含む出力プログラムに変換する命令配置手段とを備えることを特徴とする。

[0017] これにより、例えば図2に示すように、図1(a)に示すようなループ処理を二重ループ化し、最内ループの外側にプリフェッチ命令を挿入することができる。これにより、無駄なプリフェッチ実行がなくなる。よって、処理速度が向上する。また、あるdpref命令が実行されてから次のdpref命令が実行されるまでの間に、主記憶からキャッシュへのデータ転送にかかるレイテンシを隠蔽することが可能になり、インターロックが生じにくくなる。

[0018] すなわち、本発明によると、ループを二重化することにより、内側ループの外側でインターロックを起こす可能性のある命令を実行するようにすれば、当該命令を無駄に発行せずに、プログラム実行時の処理速度を向上させることができる。

[0019] また、ループを二重化することにより、インターロックを起こす可能性のある命令を発行してから次のインターロックを起こす可能性のある命令までの間のサイクル数を確保することができる。このため、プログラム実行時にインターロックを引き起こしにくくなる。

[0020] なお、プログラム変換装置は、コンパイラ、OS(Operating System)、またはCPU等の集積回路として実現可能である。

[0021] 応答待ち命令には、上述したdpref命令のようにインターロックを起こす可能性のある命令や、命令実行時に外部からの所定の応答を待つ命令の他に、応答を待つ場合と待たない場合とがある命令も含む。

[0022] なお、本発明は、このような特徴的な手段を備えるプログラム変換装置として実現することができるだけでなく、プログラム変換装置が備える特徴的な手段をステップとするプログラム変換方法として実現したり、プログラム変換装置としてコンピュータを機能させるプログラムとして実現したりすることもできる。そして、このようなプログラムは、CD－ROM(Compact Disc-Read Only Memory)等の記録媒体やインターネット等の伝送媒体を介して流通させることができるのは言うまでもない。

発明の効果

[0023] 本発明によると、プログラム実行時の処理速度を向上させることができる。

また、プログラム実行時にインターロックを引き起こしにくくなる。

図面の簡単な説明

[0024] [図1]図1は、従来の最適化技術の問題点を説明するための図である。

[図2]図2は、本発明によるループ処理の構造変換を説明するための図である。

[図3]図3は、本実施の形態に係るコンパイラシステムの構成を示す図である。

[図4]図4は、コンパイラの構成を示す図である。

[図5]図5は、コンパイラが実行する処理のフローチャートである。

[図6]図6は、ループ構造変換処理の詳細を説明するための図である。

[図7]図7は、コピー型内側ループ分割処理の詳細を示すフローチャートである。

[図8]図8は、条件型内側ループ分割処理の詳細を示すフローチャートである。

[図9]図9は、プリフェッチ命令配置処理の詳細を示すフローチャートである。

[図10]図10は、プリフェッチ命令挿入処理の詳細を示すフローチャートである。

[図11]図11は、ピーリングが必要ない場合のシンプルループ分割処理について説明するための図である。

[図12]図12は、ピーリングが必要のない場合のソースプログラムの一例を示す図である。

[図13]図13は、図12に示したソースプログラムに対応する中間言語のプログラムを示す図である。

[図14]図14は、図13に示された中間言語のプログラムを二重ループに構造変換した後の中間言語のプログラムを示す図である。

[図15]図15は、図14に示された中間言語のプログラムにプリフェッチ命令を挿入した後の中間言語のプログラムを示す図である。

[図16]図16は、ピーリングが必要な場合のシンプルループ分割処理について説明するための図である。

[図17]図17は、ループ内に複数の配列アクセスが存在する場合のループ分割処理について説明するための図である。

[図18]図18は、ループ内に複数の配列アクセスが存在する場合のループ分割処理について説明するための図である。

[図19]図19は、ループ内に複数の配列アクセスが存在し、かつ配列の要素のサイズがすべて同じではない場合のループ分割処理について説明するための図である。

[図20]図20は、ループ内に複数の配列アクセスが存在し、かつ配列の要素のサイズがすべて同じではない場合のループ分割処理について説明するための図である。

[図21]図21は、ループ内にストライドが異なる複数の配列アクセスが存在する場合のループ分割処理について説明するための図である。

[図22]図22は、ループ回数が不定なループ処理のループ分割処理を説明するための図である。

[図23]図23は、ループ回数が不定なループ処理のループ分割処理を説明するための他の図である。

[図24]図24は、ループ分割が不要な場合の最適化処理を説明するための図である。

[図25]図25は、ループ内でアクセスされる要素が主記憶上で適切にアラインされていない場合のループ分割処理を説明するための図である。

[図26]図26は、ループ内でアクセスされる要素が主記憶上で適切にアラインされていない場合のループ分割処理を説明するための図である。

[図27]図27は、アラインされていない配列要素を動的に特定して、ループ処理の最適化を行う処理について説明するための図である。

[図28]図28は、アラインされていない配列要素を説明するための図である。

[図29]図29は、アラインされていない配列要素をプロファイル情報を用いて特定し、

ループ処理の最適化を行う処理について説明するための図である。

[図30]図30は、最内ループ以外のループに対する構造変換について説明するための図である。

[図31]図31は、プラグマ「＃pragma　__loop__tiling__dpref　変数名[, 変数名]」により変数が指定された場合の最適化処理について説明するための図である。

[図32]図32は、PreTouch命令挿入時におけるピーリングが必要ない場合のシンプルループ分割処理について説明するための図である。

[図33]図33は、PreTouch命令挿入時におけるピーリングが必要な場合のシンプルループ分割処理について説明するための図である。

[図34]図34は、アラインされていない配列要素を動的に特定して、ループ処理の最適化を行う処理について説明するための図である。

符号の説明

[0025]

141 ソースプログラム

142 キャッシュパラメータ

143 アセンブラファイル

144 オブジェクトファイル

145 実行プログラム

146 実行ログデータ

147 プロファイルデータ

148 コンパイラシステム

149 コンパイラ

150 アセンブラ

151 リンカ

152 シミュレータ

153 プロファイラ

181 最適化補助情報

182 構文解析部

183 最適化情報解析部

184 一般最適化部

185 命令スケジューリング部

186 ループ構造変換部

187 命令最適配置部

188 コード出力部

発明を実施するための最良の形態

[0026] [システム構成]

図3は、本実施の形態に係るコンパイラシステムの構成を示す図である。コンパイラシステム148は、C言語等の高級言語で記述されたソースプログラム141を機械語の実行プログラム145に変換するソフトウェアシステムであり、コンパイラ149と、アセンブラ150と、リンカ151とを含む。

[0027] コンパイラ149は、キャッシュを備えるコンピュータのCPU(Central Processing Unit)をターゲットプロセッサとし、ソースプログラム141をアセンブラ言語で記述されたアセンブラファイル143に変換するプログラムである。コンパイラ149は、ソースプログラム141をアセンブラファイル143に変換する際に、キャッシュのラインサイズやレイテンシサイクル等に関する情報であるキャッシュパラメータ142や、後述するプロファイルデータ147に基づいて、最適化処理を行い、アセンブラファイル143を出力する。

[0028] アセンブラ150は、アセンブラ言語で記述されたアセンブラファイル143を機械語で記述されたオブジェクトファイル144に変換するプログラムである。リンカ151は、複数のオブジェクトファイル144を結合し、実行プログラム145を生成するプログラムである。

[0029] 実行プログラム145の開発ツールとして、シミュレータ152およびプロファイラ153が用意されている。シミュレータ152は、実行プログラム145をシミュレートし、実行時の各種実行ログデータ146を出力するプログラムである。プロファイラ153は実行ログデータ146を解析し、プログラムの実行順序等を解析したプロファイルデータ147を出力するプログラムである。

[0030] [コンパイラの構成]

図4は、コンパイラの構成を示す図である。コンパイラ149は、構文解析部182と、最適化情報解析部183と、一般最適化部184と、命令スケジューリング部185と、ループ構造変換部186と、命令最適配置部187と、コード出力部188とを含む。各構成処理部は、プログラムとして実現される。

[0031] 構文解析部182は、ソースプログラム141を入力として受け、構文解析処理を行った後、中間言語のプログラムを出力する処理部である。

[0032] 最適化情報解析部183は、キャッシュパラメータ142、プロファイルデータ147、コンパイルオプションおよびプラグマなどの中間言語の最適化処理に必要な情報を読み込み、解析する処理部である。一般最適化部184は、中間コードに一般的な最適化処理を施す処理部である。命令スケジューリング部185は、命令の並びを最適化し、命令スケジューリングを行う処理部である。コンパイルオプションおよびプラグマはいずれもコンパイラに対する指示である。

[0033] ループ構造変換部186は、一重ループを二重ループに変換する処理部である。命令最適配置部187は、変換された二重ループ内にプリフェッチ命令を配置する処理部である。コード出力部188は、最適化された中間言語仕様のプログラムをアセンブラ言語で記述されたプログラムに変換してアセンブラファイル143を出力する処理部である。

[0034] [処理の流れ]

次に、コンパイラ149の実行する処理の流れについて説明する。図5は、コンパイラ149が実行する処理のフローチャートである。

[0035] 構文解析部182は、ソースプログラム141の構文解析を行い、中間コードを生成する(S1)。最適化情報解析部183は、キャッシュパラメータ142、プロファイルデータ147、コンパイルオプションおよびプラグマなどを解析する(S2)。一般最適化部184は、最適化情報解析部183における解析結果に従い、一般的な中間コードの最適化を行う(S3)。命令スケジューリング部185は、命令のスケジューリングを行う(S4)。ループ構造変換部186は、中間コードに含まれるループ構造に着目し、必要であれば一重ループ構造を二重ループ構造に変換する(S5)。命令最適配置部187は、ループ構造内で参照されるデータをプリフェッチする命令を中間コードに挿入する(S6

)。コード出力部188は、中間コードをアセンブラコードに変換し、アセンブラファイル143として出力する(S7)。

[0036] 構文解析処理(S1)、最適化情報解析処理(S2)、一般的な最適化処理(S3)、命令スケジューリング処理(S4)およびアセンブラコード出力処理(S7)は、一般的な処理と同様であるため、その詳細な説明はここでは繰返さない。

[0037] 以下、ループ構造変換処理(S5)およびプリフェッチ命令配置処理(S6)について詳細に説明する。

[0038] 図6は、ループ構造変換処理(図5のS6)の詳細を説明するための図である。ループ構造変換部186は、ループ回数が即値で与えられており算出可能であるか、それ以外の変数等で与えられており算出不可能であるかを判断する(S11)。すなわち、ループ回数が固定であるか不定であるかを判断する。

[0039] ループ回数が不定の場合には(S11でNO)、プラグマまたはコンパイルオプションにより最低のループ回数の指定があるかまたはプログラム実行時に動的にループ回数を判定し、ループ分割をする旨の指定があるかについて判断する(S12)。

[0040] いずれかの指定がある場合(S12でYES)またはループ回数が固定値の場合には(S11でYES)、ループ内で参照されている配列の添え字が解析可能か否かについて調べる(S13)。すなわち、ループカウンタがある規則性を持って変化している場合には解析可能であると判断される。例えば、ループカウンタの値がイタレーション内で書換えられるような場合には、解析不可能であると判断される。

[0041] 添え字が解析可能である場合には(S13でYES)、ループ処理内で参照される各配列について1イタレーションで参照される要素のバイト数を求め、そのうち最小の値LBを導出する(S14)。

[0042] 次に、キャッシュのラインサイズCSを値LBで割った値が1よりも大きいか否か判断する(S15)。CS／LBの値が1よりも大きい場合には(S15でYES)、ループ処理の配列がアラインされているか否かを調べる(S16)。配列がアラインされているか否かの判断は、プラグマやコンパイルオプション等によりアラインされているとの指示があるか否かにより判断される。

[0043] 配列がアラインされていない場合には(S17でNO)、「LB＊LC／IC」がCSよりも

大きいか否かについて判断する(S16)。ここで、LCは、レイテンシのサイクル数を示し、ICは1イタレーションあたりのサイクル数を示す。「LC／IC」は、ループを複数の最内ループに分割した場合の各ループのループ回数を示しており、「LB＊LC／IC」は、各ループでのアクセス容量を示している。

[0044] 「LB＊LC／IC」がラインサイズCSよりも大きい場合には、(S16でYES)、分割後の各ループ処理では1ラインサイズ以上の要素の参照が行われる。このため、分割要因をサイクルとし、各ループ処理を二重ループ化した際の最内ループのループ回数DTを次式(1)に従い導出する(S18)。

[0045] $$DT=(LC-1)/IC+1 \quad \cdots(1)$$

「LB＊LC／IC」がラインサイズCS以下の場合(S16でNO)または配列がアラインされている場合には(S17でYES)、分割要因をサイズとし、各ループ処理を二重ループ化した際の最内ループのループ回数DTを次式(2)に従い導出する(S19)。

[0046] $$DT=(CS-1)/LB+1 \quad \cdots(2)$$

最内ループのループ回数DTが導出処理(S18またはS19)後、最内ループのループ回数DTが1よりも大きいか否か判断される(S20)。DTが1の場合には(S20でNO)、最内ループのループ回数DTが1回であるため、ループを二重ループに構造変換する必要がない。このため、ループ構造変換処理(S5)を終了させる。

[0047] 最内ループのループ回数DTが2以上の場合には(S20でYES)、ループを二重ループに構造変換した場合の外側のループ構造が作成される(S21)。外側ループ構造を生成する際に、ピーリング処理が必要か否かを判断する(S22)。ピーリング処理およびピーリング処理が必要か否かの判断方法については後述する。

[0048] ピーリング処理が必要な場合には(S22でNO)、ピーリング処理を行い、ピーリングコードを生成する(S24)。その後、コンパイルオプション「－O」または「－Os」による指定があるか否かを調べる(S25)。ここで、コンパイルオプション「－O」は、プログラムサイズおよび実行処理速度ともに平均的なアセンブラコードをコンパイラに出力させるための指示である。コンパイルオプション「－Os」は、プログラムサイズ抑制を重視したアセンブラコードをコンパイラに出力させるための指示である。

[0049] ピーリング処理する必要がないか(S22でYES)またはコンパイルオプション「－O」

または「－Os」の指定がない場合には（S25でNO）、内側ループ（最内ループ）のループ回数の条件式を生成する（S23）。

[0050] コンパイルオプション「－O」または「－Os」の指定がある場合には（S25でYES）、ピーリングされたループ処理を二重ループに畳み込み、最内ループのループ回数の条件式を生成する（S26）。

[0051] 最内ループのループ回数条件生成処理（S23、S26）の後、最内ループにおける参照の対象配列は1つであるか否か調べられる（S27）。最内ループにおける参照の対象配列が1つの場合には、（S27でYES）、ループ構造変換処理（S5）を終了する。

[0052] 最内ループにおける参照の対象配列が2つ以上ある場合には（S27でNO）、最内ループの分割個数を導出し、分割後の各最内ループのループ回数の比率を決定する（S28）。その後、分割後の最内ループ回数DTを分割個数で割った値が1よりも大きいか否か判断する（S29）。すなわち、当該値が1以下の場合には（S29でNO）、分割後の各ループ回数が1回以下であるため、分割する意味がない。このため、ループ構造変換処理（S5）を終了させる。

[0053] 当該値が1よりも大きい場合には（S29でYES）、分割後の各ループ回数が2回以上である。この場合には、コンパイルオプション「－O」または「－Ot」による指定があるか否かを調べる（S30）。コンパイルオプション「－Ot」は、実行処理速度向上を重視したアセンブラコードをコンパイラに出力させるための指示である。

[0054] コンパイルオプション「－O」または「－Os」による指定がある場合には（S30でYES）、後述する実行処理速度向上を重視したコピー型内側ループ分割処理（S31）を実行し、ループ構造変換処理（S5）を終了する。

[0055] コンパイルオプション「－O」または「－Os」による指定がない場合には（S30でNO）、後述するプログラムサイズ抑制を重視した条件型内側ループ分割処理（S32）を実行し、ループ構造変換処理（S5）を終了する。

[0056] 図7は、コピー型内側ループ分割処理（図6のS31）の詳細を示すフローチャートである。

[0057] 最内ループのループ回数DTを分割個数で割った値を細分割後内側ループ回数

とする(S41)。次に、内側ループを分割個数分だけ複製し、生成する(S42)。その後、細分割後の各内側ループ回数を細分割後内側ループ回数に修正する(S43)。さらに、DTを分割個数で割った剰余を細分割後の先頭ループのループ回数に加算し(S44)、コピー型内側ループ分割処理を終了する。

[0058] 図8は、条件型内側ループ分割処理(図6のS32)の詳細を示すフローチャートである。

[0059] 最内ループのループ回数DTを分割個数で割った値を細分割後内側ループ回数とする(S51)。次に、内側ループ回数条件の切換えswitchテーブルを生成する(S52)。すなわち、内側ループ回数を順次切り替えるように、C言語で言うところのswitch文を生成する。なお、if文であってもよい。

[0060] テーブル生成後、細分割後の各内側ループ回数条件を細分割後の内側ループ回数に修正する(S53)。その後、DTを分割個数で割った剰余を細分割後の先頭ループの回数条件に加算し(S54)、条件型内側ループ分割処理を終了する。

[0061] 図9は、プリフェッチ命令配置処理(図5のS6)の詳細を示すフローチャートである。

[0062] プリフェッチ命令配置処理では、すべてのループについて以下の処理を繰返す(ループA)。まず、着目しているループが命令挿入対象のループであるか否かを調べる(S61)。命令挿入対象のループであるか否かの情報は、ループ構造変換部186の解析結果より取得される。

[0063] 命令挿入対象のループの場合には(S61でYES)、そのループに対して条件型ループ分割が行われているか否かを調べる(S62)。条件型ループ分割が行われていれば、各条件文における命令挿入位置を解析し(S63)、プリフェッチ命令を挿入する(S64)。命令挿入対象のループに対して条件型ループ分割が行われていなければ(S62でNO)、そのループに対してコピー型ループ分割が行われているか否かを調べる(S65)。コピー型ループ分割が行われていれば(S65でYES)、そのループの手前の命令挿入位置を解析する(S66)。その後、プリフェッチ命令が挿入される(S67)。ピーリングされたループの場合には(S68でYES)、当該ループの手前に命令挿入するように命令挿入位置が解析され(S69)、その位置にプリフェッチ命令が挿入される(S70)。

[0064] 図10は、プリフェッチ命令挿入処理(図9のS64、S67およびS70)の詳細を示すフローチャートである。

[0065] 命令挿入処理では、挿入命令、挿入位置、挿入アドレス等からなる情報リストがすべて空になるまで以下を繰返す(ループB)。

[0066] プリフェッチ命令を挿入しようとしている配列要素がアライン済みであるか否かを判断する(S72)。アラインされていなければ(S72でNO)、サイクル要因に従ってループ分割されたものであるのか、サイズ要因に従ってループ分割されたものであるのかを調べる(S73)。

[0067] アライン済みであるか(S72でYES)またはサイクル要因でループ分割されたものであれば(S73でYES)、1ライン先のデータをプリフェッチする命令を挿入する(S74)。アラインされておらず、かつサイズ要因でループ分割されたものであれば(S73でNO)、2ライン先のデータをプリフェッチする命令を挿入する(S75)。最後に、解析済みの情報を情報リストから削除する(S76)。

[0068] [コンパイルオプション]

コンパイラシステム148では、コンパイラに対するコンパイルオプションとして、オプション「－fno－loop－tiling－dpref」が用意される。このオプションが指定されれば、プラグマの指定に関わらず、ループに対する構造変換は行わない。本オプションの指定がなければ、構造変換の実施はプラグマ指定の有無に従う。

[0069] [プラグマ指定]

本指定は、直後のループに対するものである。

[0070] プラグマ「＃pragma __loop__tiling__dpref 変数名[, 変数名]」により変数が指定された場合には、プラグマ指定された変数のみに着目してループ分割を行う。指定する変数は、配列でも、ポインタでもよい。

[0071] プラグマ「＃pragma __loop__tiling__dpref__all」によりループが指定された場合には、ループ内で参照される配列の全てに着目して構造変換が行われる。

[0072] 以下、いくつかの具体的曲面におけるループ分割処理について説明する。なお、以降の処理では、説明の簡単化のためC言語によるプログラム記述を行っているが、実際には中間言語による最適化処理が行われる。

[0073] [シンプルループ分割]

図11は、ピーリングが必要ない場合のシンプルループ分割処理について説明するための図である。

[0074] 図11(a)に示すようなソースプログラム282が入力された場合について考える。このソースプログラム282では、配列Aの要素が順次参照され、変数sumに加算される。ここで、配列Aの各要素のサイズは4バイトであるものとし、キャッシュの1ラインサイズは128バイト(以降の説明でも、キャッシュのラインサイズは128バイトであるものとする。)であるものとする。すなわち、キャッシュの1ラインには配列Aの要素が32個記憶される。また、ソースプログラム282に含まれるループのイテレーションの回数128回は、32の整数倍である。このため、ソースプログラム282は、図11(b)のプログラム284に示すように、二重ループに構造変換することができる。すなわち、最内ループでは32回の繰り返し処理を行い、その外のループでは、最内ループを4回繰返すループ処理を行う。最内ループ処理ではキャッシュの1ライン分のデータが参照される。その後、図11(c)のプログラム286に示されるように、最内ループの実行前に、プリフェッチ命令(dpref(&A[i+32]))が挿入される。プリフェッチ命令を挿入することにより、最内ループ実行時には、当該ループで参照される配列Aの要素がキャッシュに乗っていることになる。

[0075] 図12～図15は、ピーリングが不要なシンプルループ分割処理における中間言語の推移を説明するための図である。

[0076] 図12は、図11(a)と同様に、ピーリングが必要のない場合のソースプログラムの一例を示す図である。図13は、図12に示したソースプログラム240に対応する中間言語のプログラムである。[BGNBBLK]と[ENDBBLK]とで挟まれた内部の命令列が1つの基本ブロックに対応しており、[BGNBBLK]B1で始まる基本ブロックがforループの直前までの処理を示しており、[BGNBBLK]B2で始まる基本ブロックがforループを示しており、[BGNBBLK]B3で始まる基本ブロックがforループの後の処理を示している。

[0077] 図14は、図13に示された中間言語のプログラムを二重ループに構造変換した後の中間言語のプログラムを示している。[BGNBBLK]B2で始まる基本ブロックが最内

ループに対応しており、[BGNBBLK]B4および[BGNBBLK]B5で始まるループがその外側のループに対応している。

[0078] 図15は、図14に示された中間言語のプログラムにプリフェッチ命令を挿入した後の中間言語のプログラムを示している。プログラム270では、[BGNBBLK]B4で始まる基本ブロックの内部にプリフェッチ命令(dpref)が新たに挿入されている。

[0079] 図16は、ピーリングが必要な場合のシンプルループ分割処理について説明するための図である。

[0080] 図16(a)に示すようなソースプログラム292が入力された場合について考える。このソースプログラム292では、配列Aの要素が順次参照され、変数sumに加算される。ここで、配列Aの各要素のサイズは4バイトであるものとする。すなわち、キャッシュの1ラインには配列Aの要素が32個記憶される。また、ソースプログラム292に含まれるループのイテレーションの回数は140回であるものとする。すなわち、1ラインに記憶される配列Aの要素数32で割った場合に余りが出る数である。

[0081] このような場合には、図16(b)に示すプログラム294のように、140を32で割った余りのループ回数をピーリングし、それ以外の部分を図11(b)と同様に二重ループ構造に構造変換する。その後、ピーリングされた部分を二重ループ構造に含ませるためのピーリング畳み込み処理を行い、図16(c)に示すようなプログラム296が得られる。すなわち、通常状態では最内ループで32回の繰り返し処理が行われ、最後に最内ループが実行される場合には、残りの12(＝140－128)回の繰り返し処理が行われる。その後、図16(d)のプログラム298に示されるように、最内ループの実行前に、プリフェッチ命令(dpref(&A[i＋32]))が挿入される。

[0082] [複数配列アクセスが存在する場合(ピーリング必要なし)]

図17は、ループ内に複数の配列アクセスが存在する場合のループ分割処理について説明するための図である。

[0083] 図17(a)に示すようなソースプログラム301が入力された場合について考える。このソースプログラム301では、配列Aおよび配列Bの要素が順次参照され、当該要素同士の積が変数sumに加算される。ここで、配列Aおよび配列Bの各要素はそれぞれ4バイトであるものとする。すなわち、キャッシュの1ラインには配列Aの要素が32個記

憶される。または、配列Bの要素が32個記憶される。すなわち、1ラインに格納される要素数は配列Aと配列Bとで同じである。また、ソースプログラム301に含まれるループのイテレーションの回数128回は、32の整数倍である。このため、ソースプログラム301は、図17(b)のプログラム302に示すように、ピーリングをすることなく二重ループに構造変換することができる。

[0084] 複数配列アクセスが存在する場合の二重ループ構造は、コピー型と呼ばれる実行処理速度を向上させるための最適化と、条件型と呼ばれるプログラムサイズを小さくするための最適化との二種類がある。

[0085] まず、コピー型の最適化について説明する。プログラム302に含まれる最内ループのループ回数を配列Aと配列Bとの要素の大きさの比で分割する。ここでは、配列Aと配列Bとはともに同じ要素の大きさである。したがって、図17(c)に示すプログラム303のように最内ループを二等分し、ループ回数が16回の最内ループ2つに分割する。次に、図17(d)のプログラム304に示すように、各最内ループの直前にプリフェッチ命令を挿入する。最初の最内ループの直前には、1ライン分の配列Aの要素をプリフェッチするためのプリフェッチ命令(dpref(&A[i+32]))が挿入され、2番目の最内ループの直前には、1ライン分の配列Bの要素をプリフェッチするためのプリフェッチ命令(dpref(&B[i+32]))が挿入される。

[0086] このようにプリフェッチ命令間にループ処理を挿入させることにより、異なる配列に対するプリフェッチ命令が連続することが無くなり、プリフェッチ命令実行によるレイテンシを隠蔽することができる。このため、実行処理速度を向上させることができる。

[0087] 次に、条件型の最適化について説明する。条件型の場合も、コピー型の場合と同様にして最内ループのループ回数を配列Aと配列Bとの要素の大きさの比で分割する。ただし、プログラム303のように最内ループを2つ並べるのではなく、図17(e)に示すプログラム305のように最内ループの個数は1つであり、そのループ回数を条件分岐させるようにしている。すなわち、変数K=1の場合と、K=0の場合とで最内ループのループ回数Nを変えるようにしている。ただし、この例では変数Kの値に関係なく最内ループの回数Nは16回となっている。次に、図17(f)に示すプログラム306のように、K=1の場合には配列Aの要素を1ライン分プリフェッチし、K=0の場合に

は配列Bの要素を1ライン分プリフェッチするように条件分岐式およびプリフェッチ命令の挿入が行われる。なお、ここでは、最適化によりループ回数Nは即値16に置き換えられている。

[0088] このように、最内ループの個数を1つにし、条件分岐式で最内ループのループ回数およびプリフェッチ命令を変えるようにすることにより、最終的に生成される機械語命令のプログラムサイズを小さくすることができる。ただし、条件分岐処理があるため、コピー型に比べて処理速度が多少遅くなる可能性がある。

[0089] [複数配列アクセスが存在する場合(ピーリング必要)]

図18は、ループ内に複数の配列アクセスが存在する場合のループ分割処理について説明するための図である。

[0090] 図18(a)に示すようなソースプログラム311が入力された場合について考える。このソースプログラム311では、配列Aおよび配列Bの要素が順次参照され、当該要素同士の積が変数sumに加算される。ここで、配列Aおよび配列Bの各要素はそれぞれ4バイトであるものとする。すなわち、キャッシュの1ラインには配列Aの要素が32個記憶される。または、配列Bの要素が32個記憶される。すなわち、1ラインに格納される要素数は配列Aと配列Bとで同じである。また、ソースプログラム311に含まれるループのイタレーションの回数は140回であるものとする。

[0091] したがって、ソースプログラム311を二重ループに構造変換する場合には、図16(b)に示したプログラム294と同様、図18(b)に示すようにピーリング処理されたプログラム312が生成される。

[0092] コピー型の最適化を行う際には、配列Aと配列Bとの要素の大きさの比で最内ループを分割する。すると、図18(c)に示すプログラム313が生成される。次に、図18(d)のプログラム314に示すように、最初の最内ループの直前には、1ライン分の配列Aの要素をプリフェッチするためのプリフェッチ命令(dpref(&A[i+32]))が挿入され、2番目の最内ループの直前には、1ライン分の配列Bの要素をプリフェッチするためのプリフェッチ命令(dpref(&B[i+32]))が挿入される。なお、ピーリング処理された最終ループの直前にはプリフェッチ命令は挿入されない。これは、その前の二重ループ処理におけるプリフェッチ命令実行により所望のデータがキャッシュにプリフェ

ッチされているからである。

[0093] 条件型の最適化を行う際には、プログラム312に対してピーリング畳み込み処理を行い、図18(e)に示されるようなプログラム315を得る。ピーリング畳み込み処理は、図16を参照して説明したものと同様である。次に、最内ループのループ回数を配列Aと配列Bとの要素の大きさの比で分割し、当該ループ回数を条件分岐させるように図18(f)に示すプログラム316を作成する。プログラム316においては、変数Kの値を交互に変更させ、変数Kの値に対応するようにループカウンタNの値を変化させる。次に、図18(g)のプログラム317に示すように、Kの値の変化に伴い、配列Aおよび配列Bの要素を1ライン分ずつ交互にプリフェッチするように、条件分岐式中にプリフェッチ命令を挿入する。

[0094] このように、ピーリングが必要な場合であっても、コピー型の場合にはピーリングの部分を二重ループとは別のループにし、条件型の場合には、条件分岐式によりピーリングの場合のループカウンタの回数を変えるようにすることにより、ループ内に複数の配列アクセスがあり、かつピーリングが必要な場合であっても、プリフェッチによるレイテンシを考慮した最適化を行うことができる。

[0095] [サイズが異なる複数配列アクセスが存在する場合(ピーリング必要なし)]

図19は、ループ内に複数の配列アクセスが存在し、かつ配列の要素のサイズがすべて同じではない場合のループ分割処理について説明するための図である。

[0096] 図19(a)に示すようなソースプログラム321が入力された場合を考える。ここで、配列Aの要素は4バイト、配列Bの要素は2バイトとする。すなわち、キャッシュの1ラインには配列Aの要素が32個、配列Bの要素が64個記憶される。

[0097] この場合、要素サイズの小さい配列Bに着目し、配列Bの要素に応じたループの構造変換を行う。すなわち、図19(b)のプログラム322のように、最内ループのループ回数を1ラインに収まるキャッシュBの要素数64にし、二重ループに構造変換する。最内ループでは、配列Bに関しては1ライン分の要素が消費されるが、配列Aに関しては2ライン分の要素が消費されることになる。このため、最内ループ処理を実行するためには合計3ライン分のデータが必要になる。

[0098] このため、コピー型の最適化を行う際には、図19(c)のプログラム323に示すように

、最内ループを3つに分割し、図19(d)のプログラム324に示すように、各最内ループの直前にプリフェッチ命令を挿入する。ここでは、1番目の最内ループの直前には、2ライン先の配列Aの要素をプリフェッチするプリフェッチ命令(dpref(&A[i+64]))を挿入し、2番目の最内ループの直前には3ライン先の配列Aの要素をプリフェッチするプリフェッチ命令(dpref(&A[i+96]))を挿入し、3番目の最内ループの直前には1ライン先の配列Bの要素をプリフェッチするプリフェッチ命令(dpref(&B[i+64]))を挿入している。また、3つの最内ループのループ回数を処理順に22、21および21としている。これは、最外ループの条件分岐判断が3番目の最内ループ実行後に行われるため、3番目の最内ループのループ回数を少なくすることにより、全体としての処理速度を向上させるためである。

[0099] また、条件型の最適化を行う際には、図19(e)のプログラム325に示すように、1回の最内ループ処理につき、変数Kの値を0から2までの範囲内で更新させ、変数Kの値による条件分岐処理により最内ループのループ回数Nを22、21および21のうちのいずれかに設定する。その後、ループ回数Nの最内ループを実行させる。次に、図19(f)のプログラム326に示すように、変数Kの値が0の場合にはプリフェッチ命令(dpref(&A[i+64]))を実行させ、変数Kの値が1の場合にはプリフェッチ命令(dpref(&A[i+96]))を実行させ、変数Kの値が2の場合にはプリフェッチ命令(dpref(&B[i+64]))を実行させるように最適化を行う。

[0100] [サイズが異なる複数配列アクセスが存在する場合(ピーリング必要)]

図20は、ループ内に複数の配列アクセスが存在し、かつ配列の要素のサイズがすべて同じではない場合のループ分割処理について説明するための図である。

[0101] 図20(a)に示すソースプログラム331は、図19(a)に示したソースプログラム321とループ回数が異なるのみである。したがって、ソースプログラム321と同様、配列Aの要素は4バイト、配列Bの要素は2バイトである。図20(b)に示すように、ソースプログラム321のループを二重ループに構造変換し、ループ回数140を配列Bの1ライン分の要素数64で割った余りをピーリング処理すると、プログラム322が得られる。コピー型の最適化処理を行う場合には、図19(c)および図19(d)を参照して説明したように、二重ループの最内ループを3分割し、プリフェッチ命令を挿入することにより、図20

(c)に示すプログラム333が得られる。条件型の最適化処理を行う場合には、図19(e)および図19(f)を参照して説明したように、条件分岐式によりループ回数およびプリフェッチ命令を制御し、最終的に図20(e)に示すプログラム335が得られる。

[0102] [ストライドが異なる複数配列アクセスが存在する場合]

図21は、ループ内にストライドが異なる複数の配列アクセスが存在する場合のループ分割処理について説明するための図である。

[0103] ストライドとは、ループ処理における配列要素の増分値(アクセス幅)のことを示す。図21(a)に示すようなソースプログラム341が入力された場合を考える。ここで、配列Aの要素および配列Bの要素はともに4バイトであるものとする。ソースプログラム341では、ループのイタレーションごとに、配列Aの要素は1ずつ増加するのに対し、配列Bの要素は2ずつ増加する。すなわち、配列Bのアクセス幅は配列Aのアクセス幅の2倍である。最小アクセス幅の配列Aに着目すると、1ラインには配列Aの要素が32個収まる。このため、最内ループのループ回数を32回とした二重ループへの構造変換を行うと、図21(b)に示すプログラム342が得られる。最内ループでは、配列Aに関しては1ライン分の要素が消費されるが、配列Bに関しては2ライン分の要素が消費されることになる。このため、最内ループ処理を実行するためには合計3ライン分のデータが必要になる。

[0104] よって、コピー型の最適化を行う際には、図21(c)のプログラム343に示すように、最内ループを3つに分割し、図21(d)のプログラム344に示すように、各最内ループの直前にプリフェッチ命令を挿入する。ここでは、1番目の最内ループの直前には、1ライン先の配列Aの要素をプリフェッチするプリフェッチ命令(dpref(&A[i+32]))を挿入し、2番目の最内ループの直前には2ライン先の配列Bの要素をプリフェッチするプリフェッチ命令(dpref(&B[i*2+64]))を挿入し、3番目の最内ループの直前には3ライン先の配列Bの要素をプリフェッチするプリフェッチ命令(dpref(&B[i*2+96]))を挿入している。

[0105] また、条件型の最適化を行う際には、図21(e)のプログラム345に示すように、1回の最内ループ処理につき、変数Kの値を0から2までの範囲内で更新させ、変数Kの値による条件分岐処理により最内ループのループ回数Nを11、11および10のうち

のいずれかに設定する。その後、ループ回数Nの最内ループを実行させる。次に、図21(f)のプログラム346に示すように、変数Kの値が0の場合にはプリフェッチ命令(dpref(&A[i+32]))を実行させ、変数Kの値が1の場合にはプリフェッチ命令(dpref(&B[i*2+64]))を実行させ、変数Kの値が2の場合にはプリフェッチ命令(dpref(&B[i*2+96]))を実行させるように最適化を行う。

[0106] [ループ回数が不定な場合]

図22は、ループ回数が不定なループ処理のループ分割処理を説明するための図である。

[0107] 図22(a)に示すソースプログラム351が入力された場合を考える。ソースプログラム351に含まれるループ回数は変数Valにより特定され、コンパイル時には不定である。しかし、最低128回は繰り返し処理が行われることがプラグマ指定「#pragma __min__iteration=128」により、保証されている。ここで、配列Aは4バイトであるものとする。すなわち、キャッシュの1ラインには配列Aの要素が32個記憶される。

[0108] プラグマ指定に従い、ループ処理を最初の128回のループ処理と、それ以降の変数Valで特定されるループ回数のループ処理とに分割し、それぞれをシンプルループの場合と同様に二重ループ化すると図22(b)に示すプログラム352が得られる。

[0109] コピー型の最適化処理を行う場合には、プログラム352の最内ループの直前に1ライン先の配列Aの要素をプリフェッチするためのプリフェッチ命令(dpref(&A[i+32]))を挿入することにより、図22(c)に示すプログラム353が得られる。

[0110] 条件型の最適化処理を行う場合には、後半のループ処理をピーリング畳み込みし、最外ループ回数が128回になるまでは、最内ループの回数を32回にし、それ以降は最内ループの回数を(Val−128)回に設定する分岐命令を挿入する。すると、図22(d)に示すようなプログラム354が得られる。

[0111] 最後に、最内ループの実行前にプリフェッチ命令(dpref(&A[i+32]))を挿入することにより図22(e)に示すようなプログラム355が得られる。

[0112] 図23は、ループ回数が不定なループ処理のループ分割処理を説明するための他の図である。

[0113] 図23(a)に示すソースプログラム361が入力された場合を考える。ソースプログラム

361に含まれるループ回数は変数Nにより特定され、コンパイル時には不定である。また、ソースプログラム361は、ソースプログラム351と異なり、最低のループ回数を示すプラグマ指定がない。

[0114] ループ回数が小さなループ処理に対してループの構造変換を行い、最適化を行ったとしても、最適化の効果が表れにくい。このため、このような場合には、最適化の効果を高めるために、ループ回数があるしきい値よりも大きければ最適化されたループ処理を実行し、それ以外の場合には通常のループ処理を実行するようにする。例えば、あるしきい値を1024とした場合には、図23(b)のプログラム362に示されるように、ループ回数Nが1024を超える場合には、最初の1024回のループ処理については二重ループを実行し、残りの回数のループ処理については、ピーリングされたループ処理を行うようにする。また、ループ回数Nが1024以下の場合には、二重ループは実行せずに、ピーリングされたループ処理を実行するようにする。その後、二重ループの最内ループの直前にプリフェッチ命令(dpref(&A[i+32]))を挿入することにより図23(c)に示すような最適化されたプログラム363が生成される。

[0115] [ループ分割が不要な場合]

図24は、ループ分割が不要な場合の最適化処理を説明するための図である。図24(a)に示すソースプログラム371が入力された場合には、ループ中で1ライン分のデータ(A[i]～A[i+31])を完全に使い切ってしまう。このような場合には、二重ループ化する必要はない。このため、図24(b)に示すプログラム372ようにループの先頭にループ内で使用されるデータの1ライン先のデータをプリフェッチするプリフェッチ命令(dpref(&A[i+32]))を挿入することにより最適化が行われる。

[0116] また、ループ内の処理サイクル数がプリフェッチ命令で必要とされる処理サイクル数よりも大きいような場合にも、ループを二重化する必要はなく、ループの先頭にプリフェッチ命令を挿入してもプリフェッチ命令のレイテンシは隠蔽することができる。

[0117] [ループ内でアクセスされる要素がアラインされていない場合]

図25および図26は、ループ内でアクセスされる要素が主記憶上で適切にアラインされていない場合のループ分割処理を説明するための図である。これまでの説明では、ループ内でアクセスされる要素が主記憶上で適切にアラインされている場合を想

定して話を進めてきた。アラインされていることが予めプラグマや、コンパイルオプションの指定によりわかっている場合には、上述の例で説明したような最適化が行われる。

[0118] しかし、一般的にはコンパイラは、それらの要素がアラインされているか否かは実行時まではわからない。このため、コンパイラは、ループ内アクセス要素が主記憶上で適切にアラインされていないことを前提として最適化を行う必要がある。

[0119] すなわち、図25(a)に示すようなソースプログラム381が与えられた場合に、配列Aの要素サイズを4バイトとすると、図11を参照して説明したシンプルループ分割と同様にして、最適化が行われる。ただし、要素がアラインされていないことを前提としているため、最内ループの前に挿入されるプリフェッチ命令(dpref(&A[i+64]))は2ライン先の配列Aの要素をプリフェッチ指定している。また、ループ処理に先立って、ループ内でアクセスされる配列の要素A[0]～A[63]を確保するために、プリフェッチ命令(dpref(&A[0])およびdpref(&A[32]))がプリフェッチのレイテンシを十分隠蔽できる位置に挿入され、図25(b)に示すようなプログラム382が生成される。

[0120] また、図26(a)に示されるようなソースプログラム391が与えられた場合には、図16と同様にして、ピーリング処理された部分のループを畳み込んだ後に、2ライン先の配列Aの要素をプリフェッチする命令(dpref(&A[i+64]))が挿入される。また、プログラム382と同様にプリフェッチ命令(dpref(&A[0])およびdpref(&A[32]))が挿入され、図26(b)に示すような最適化されたプログラム392が生成される。

[0121] [動的アライン解析コードの挿入による構造変換分割]

図27は、アラインされていない配列要素を動的に特定して、ループ処理の最適化を行う処理について説明するための図である。図27(a)に示すソースプログラム401が入力された場合を考える。ここで、配列Aの要素は4バイトであるものとする。

[0122] 配列Aの先頭アドレス(要素A[0]のアドレス)の所定のビットがキャッシュのラインを示しており、そのビット内のさらにあるビットは、ラインの先頭からのオフセットを示している。したがって、「A&Mask」というビット同士の論理演算を行うことにより、ラインの先頭からのオフセットを取り出すことができる。ここで、マスク値Maskはあらかじめ定められた値である。配列Aの先頭アドレスから取り出されたオフセット値を予め定めら

れた補正値Corだけ右シフトすることにより、配列Aの先頭要素A[0]が1ライン内で先頭から何番目に位置しているかがわかる。よって、次式(3)にしたがって、ライン上でアラインされていない要素の数nを求めることができる。

[0123] $$n=32-(A\&Mask)>>Cor \quad \cdots(3)$$

すなわち、図28に示すように、キャッシュ431にフェッチした場合に、アラインされない配列Aの要素(A[0]～A[n－1])とアラインされる配列Aの要素とが区別されることになる。

[0124] したがって、図27(b)のプログラム402に示すように、式(3)に従いアラインされていない配列Aの要素数nを求める。次に、要素数nに従って、アラインされていない配列Aの要素(A[0]～A[n－1])についてのループ処理を行う。その後、アラインされている配列Aの要素(A[n]以降の要素)については、図11に示したシンプルループ分割の場合と同様に二重ループ化を行う。

[0125] その後、ピーリングされているループ405について、畳み込み処理を行うと、図27(c)に示すようなプログラム403が生成される。また、図27(d)に示すように、プリフェッチ命令(dpref(&A[i＋32]))を挿入することにより、最適化されたプログラム404が得られる。

[0126] [プロファイル情報を用いた構造変換分割]

図29は、アラインされていない配列要素をプロファイル情報を用いて特定し、ループ処理の最適化を行う処理について説明するための図である。アラインされていない配列の要素数を図27のように計算から求めるのではなく、プロファイル情報から取得する。取得したアラインされていない配列の要素数Nに基づいて、図27に示したのと同様の処理を行い、図29(a)に示すソースプログラム411を図29(b)に示すプログラム412のように変換する。その後、ピーリングされたループ部分を畳み込み、図29(c)に示すプログラム413を得る。最後に、図29(d)に示すプリフェッチ命令を挿入することにより最適化されたプログラム414を得る。

[0127] [最内ループ以外のループに対する構造変換]

図30は、最内ループ以外のループに対する構造変換について説明するための図である。

[0128] 図30(a)に示すソースプログラム421が与えられた場合を考える。ソースプログラム421では、二重ループ処理が行われており、最内ループ処理424で参照される配列Aの要素は1バイトであるものとする。最内ループ処理424のループ回数は4回であるため、最内ループ処理424では配列Aの要素が4バイト分参照される。したがって、最内ループ処理424で参照される要素のバイト数が小さいため、このような場合には、最内ループ処理424を1つの固まりとして考え、最外ループを、図30(b)に示すプログラム422のように、二重ループに構造変換する。その後、2番目のループ処理の実行前にキャッシュの1ライン分の配列Aの要素をプリフェッチする命令(dpref(&A[j+128]))が挿入され、図30(c)に示すような最適化されたプログラム423が得られる。

[0129] [プラグマ「#pragma __loop__tiling__dpref 変数名[, 変数名]」による変数指定]

図31は、プラグマ「#pragma __loop__tiling__dpref 変数名[, 変数名]」により変数が指定された場合の最適化処理について説明するための図である。図31(a)に示すように、プラグマ「#pragma __loop__tiling__dpref b」との指定がソースプログラム中に含まれる場合には、ループ内の配列bのみに着目して構造変換が行われ、配列aは無視される。従って、図31(b)に示すような二重ループ化が実行され、配列bをプリフェッチする命令のみが挿入される。

[0130] 以上説明したように、本実施の形態に係るコンパイラシステムによると、ループ処理を二重化し、最内ループの外側でプリフェッチ命令を実行するようにしている。このため、無駄なプリフェッチ命令の発行を防ぐことができ、プログラム実行時の処理速度を向上させることができる。また、ループ処理を二重化することにより、プリフェッチ命令を実行してから次のプリフェッチ命令を実行するまでのサイクル数を確保することができる。このため、レイテンシを隠蔽し、インターロックを防ぐことができる。

[0131] 以上、本発明の実施の形態に係るコンパイルシステムについて、実施の形態に基づいて説明したが、本発明は、この実施の形態に限定されるものではない。

[0132] 例えば、命令最適配置部187で配置される命令は、プリフェッチ命令に限られず、通常のメモリアクセス命令や外部処理を起動してその処理結果を待つ命令などのよう

な応答待ち命令、実行した結果、結果的にインターロックを起こす可能性のある命令、実行後に所定の資源が参照可能になるまでに複数サイクルを要する命令などであってもよい。応答待ち命令には、常に応答を待つ命令の他に、応答を待つ場合と待たない場合とがある命令も含む。

[0133] また、キャッシュを備えないコンピュータのCPUをターゲットプロセッサとして、各種処理のレイテンシを隠蔽し、インターロックを防ぐようなコードを出力するコンパイルシステムであってもよい。

[0134] さらに、CPUで実行させる機械語命令列を逐次解釈しながら、本実施の形態で説明したループ構造変換等の処理を実行するOS(Operating System)として実現してもよい。

[0135] また、以下に示すようなPreTouch命令のように、インターロックを起こす可能性のない命令に対しても本発明は適用可能である。PreTouch命令とは、引数で指定される変数を記憶するための領域をキャッシュ上に事前に確保するのみの処理を行う命令である。以下に、ループの構造変換を行い、PreTouch命令を挿入する処理について説明する。

[0136] [シンプルループ分割]

図32は、PreTouch命令挿入時において、対象領域がキャッシュサイズでアラインされており、ピーリングが必要ない場合のシンプルループ分割処理について説明するための図である。

[0137] 図32(a)に示すようなソースプログラム502が入力された場合について考える。このソースプログラム502では、ループ回数iと変数valとの演算結果(乗算結果)を配列Aの要素に順次代入する処理を定義している。ここで、配列Aの各要素のサイズは4バイトであるものとし、キャッシュの1ラインサイズは128バイト(以降の説明でも、キャッシュのラインサイズは128バイトであるものとする。)であるものとする。すなわち、キャッシュの1ラインには配列Aの要素が32個記憶される。また、ソースプログラム502に含まれるループのイテレーションの回数128回は、32の整数倍である。

[0138] このため、ソースプログラム502は、図32(b)のプログラム504に示すように、二重ループに構造変換することができる。すなわち、最内ループでは32回の繰り返し処

理を行い、その外のループでは、最内ループを4回繰返すループ処理を行う。最内ループ処理ではキャッシュの1ライン分のデータが配列Aに代入される。その後、図32(c)のプログラム506に示されるように、最内ループの実行前に、キャッシュ領域確保命令(PreTouch(&A[i]))が挿入される。PreTouch命令を挿入することにより、最内ループ実行時には、当該ループで定義される配列Aの要素がキャッシュ領域に確保されていることになる。これにより、不要なメインメモリからのデータ転送を引き起こすことがなくなり、バス占有率を軽減することができるようになる。

[0139] 図33は、PreTouch命令挿入時におけるピーリングが必要な場合のシンプルループ分割処理について説明するための図である。

[0140] 図33(a)に示すようなソースプログラム512が入力された場合について考える。このソースプログラム512では、ループ回数iと変数valとの演算結果(乗算結果)が配列Aの要素に順次代入する処理を定義している。ここで、配列Aの各要素のサイズは4バイトでキャッシュサイズにアラインされているものとする。すなわち、キャッシュの1ラインには配列Aの要素が32個記憶される。また、ソースプログラム512に含まれるループのイテレーションの回数は140回であるものとする。すなわち、1ラインに記憶される配列Aの要素数32で割った場合に余りが出る数である。

[0141] このような場合には、図33(b)に示すプログラム514のように、140を32で割った余りのループ回数をピーリングし、それ以外の部分を図32(b)と同様に二重ループ構造に構造変換する。その後、ピーリングされた部分を二重ループ構造に含ませるためのピーリング畳み込み処理を行い、図33(c)に示すようなプログラム516が得られる。すなわち、通常状態では最内ループで32回の繰り返し処理が行われ、最後に最内ループが実行される場合には、残りの12(=140−128)回の繰り返し処理が行われる。その後、図33(d)のプログラム518に示されるように、最内ループの実行前に、キャッシュ領域確保命令(PreTouch(&A[i]))が挿入される。ただし、領域確保処理は、1ライン単位で行なわれる。このため、オブジェクトA以外の領域を確保する可能性がある最後の最内ループ実行時には、PreTouch命令を発行しないようにし、オブジェクトA以外の領域を確保しないようにする。

[0142] [動的アライン解析コードの挿入による構造変換分割]

図34は、アラインされていない配列要素を動的に特定して、ループ処理の最適化を行う処理について説明するための図である。図34(a)に示すソースプログラム522が入力された場合を考える。ここで、配列Aの要素は4バイトであるものとする。

[0143] 配列Aの先頭アドレス(要素A[0]のアドレス)の所定のビットがキャッシュのラインを示しており、そのビット内のさらにあるビットは、ラインの先頭からのオフセットを示している。したがって、「A&Mask」というビット同士の論理演算を行うことにより、ラインの先頭からのオフセットを取り出すことができる。ここで、マスク値Maskはあらかじめ定められた値である。ここでは、[Mask＝0x7F]としている。ループ初回にアクセスされる配列Aの要素のアドレスから取り出されたオフセット値を、マスク値Maskから減算を行い、予め定められた補正値Corだけ右シフトすることにより、配列Aの要素A[X]が1ライン内で先頭から何番目に位置しているかがわかる。よって、次式(4)にしたがって、ライン上でアラインされていない要素の数PRLGを求めることができる。

[0144] $$PRLG = (Mask - (\&A[X]) \,\&\, Mask) >> Cor \quad \cdots(4)$$

さらに、ループの最後に参照される配列Aの要素(A[Y－1])の次の要素(A[Y])が1ライン内で先頭から何番目に位置しているかを、次式(5)に従って求めることにより、1ラインを満たしきれていない要素の数EPLGで求めることができる。

[0145] $$EPLG = (\&A[Y]) \,\&\, Mask) >> Cor \quad \cdots(5)$$

さらに、1ライン分の処理を余ることなく行うループ回数KRNLを次式(6)に従って求めることができる。

[0146] $$KNRL = (Y-X) - (PRLG+EPLG) \quad \cdots(6)$$

すなわち、図34(b)のプログラム524に示すように、キャッシュの領域に配列Aが割当てられる場合に、アラインされない配列Aの要素(A[X]～A[X＋PRLG－1])と、アラインされかつ1ラインの倍数のサイズとなる配列Aの要素(A[X＋PRLG]～A[X＋PRLG＋KRNL－1])と、アラインされているが1ラインのサイズを満たさない配列Aの要素(A[X＋PRLG＋KRNL]～A[X＋PRLG＋KRNL＋ERLG－1])とが区別されることになる。

[0147] したがって、図34(b)のプログラム524に示すように、式(4)に従ったアラインされていない配列Aの要素数PRLGを求める処理等が行なわれる。次に、要素数PRLGに

従って、アラインされていない配列Aの要素(A[X]～A[X＋PRLG－1])についてのループ処理を行う。その後、アラインされている配列Aの要素(A[X＋PRLG]～A[X＋PRLG＋KRNL－1]の要素)については、図32(b)に示したシンプルループ分割の場合と同様に二重ループ化を行う。さらに、EPLG＞0であるならば、ピーリング処理が必要となる為、図33(b)に示したピーリング必要時の場合と同様にピーリング処理を行う。

[0148] その後、ピーリングされているループについて、畳み込み処理を行うと、図34(c)に示すようなプログラム526が生成される。また、図34(d)に示すように、キャッシュ領域確保命令(PreTouch(&A[i]))が挿入することにより、最適化されたプログラム528が得られる。

[0149] ただし、領域確保命令を挿入するのは、アラインされている領域でかつキャッシュの1ライン全てを使用する最内ループに対してのみである。

産業上の利用可能性

[0150] 本発明はインターロックを起こす可能性のある命令の発行を制御するコンパイラ、OS、プロセッサで実行されるプロセス等に適用できる。

請求の範囲

[1] 実行時に外部からの所定の応答を待つ命令を含む命令セットを備えたプロセッサ向けのプログラム変換装置であって、

入力プログラムに含まれる繰返し回数がx回のループを、繰返し回数がy回のループを内側ループとし、繰返し回数がx／y回のループを外側ループとするネスト構造への変換である二重ループ変換を行うループ構造変換手段と、

前記内側ループの外部の位置に、前記命令を配置することにより、当該命令を含む出力プログラムに変換する命令配置手段と

を備えることを特徴とするプログラム変換装置。

[2] 前記ループ構造変換手段は、

前記入力プログラムに含まれるループを検出するループ検出部と、

前記ループの繰返し回数を検出する繰返し回数検出部と、

前記命令実行時の前記所定の応答を待つサイクル数である応答待ちサイクル数を検出する応答待ちサイクル数検出部と、

前記ループの1回の繰り返し処理に要する1シーケンスサイクル数を検出する1シーケンスサイクル数検出部と、

前記ループを、繰返し回数が（前記応答待ちサイクル数／前記1シーケンスサイクル数）回であるループに分割するループ分割部と、

繰返し回数が（前記応答待ちサイクル数／前記1シーケンスサイクル数）回のループを内側ループとし、繰返し回数が（前記ループの繰返し回数／前記内側ループの繰返し回数）回をループの外側ループとするネスト構造への変換である二重ループ変換を行う二重ループ変換部と

を有することを特徴とする請求項1に記載のプログラム変換装置。

[3] さらに、最適化に関する最適化指示情報を受け取る最適化指定情報受け取り手段を備える

ことを特徴とする請求項1に記載のプログラム変換装置。

[4] 前記最適化指定情報受け取り手段は、前記入力プログラムに含まれるループの最低繰り返し回数を受け取り、

前記ループ構造変換手段は、ループの実行回数が不定な場合は、前記最低繰り返し回数に基づいて、前記最低繰り返し回数の繰り返し処理を前記ループより取り出し、取り出したループの繰り返し処理に対して二重ループ変換を行う

ことを特徴とする請求項3に記載のプログラム変換装置。

[5] 前記命令は、インターロックを発生させる可能性のある命令である

ことを特徴とする請求項1に記載のプログラム変換装置。

[6] 前記インターロックを発生させる可能性のある命令は、主記憶装置からキャッシュへのデータのプリフェッチ命令である

ことを特徴とする請求項5に記載のプログラム変換装置。

[7] さらに、命令のスケジューリングを行うスケジューリング手段を備え、

前記ループ構造変換手段は、

前記繰り返し回数がx回のループを、前記スケジューリング手段により得られた結果から、前記プリフェッチを実行するのに必要なサイクル数分だけ実行されるような繰り返し回数がy回のループに分割し、繰り返し回数がy回のループを内側ループ、繰り返し回数がx／y回のループを外側ループとするネスト構造への変換である二重ループ変換を行う

ことを特徴とする請求項6に記載のプログラム変換装置。

[8] 前記命令は、実行後に、所定の資源が参照可能状態になるまでに複数サイクルを要する命令である

ことを特徴とする請求項1に記載のプログラム変換装置。

[9] 前記複数を要する命令は、主記憶装置またはキャッシュをアクセスする命令である

ことを特徴とする請求項8に記載のプログラム変換装置。

[10] 前記ループ構造変換手段は、

前記繰り返し回数がx回のループを、当該ループ内で参照される配列のアドレスがキャッシュのラインサイズ進む分だけ実行されるような繰り返し回数がy回のループに分割し、繰り返し回数がy回のループを内側ループ、繰り返し回数がx／y回のループを外側ループとする二重ループ変換を行う

ことを特徴とする請求項1に記載のプログラム変換装置。

[11] 前記ループ構造変換手段は、前記配列が複数存在する場合に、二重ループ変換を行った前記繰り返し回数がy回のループを、さらに、前記配列の数に基づいて案分する案分変換を行う

ことを特徴とする請求項10に記載のプログラム変換装置。

[12] 前記案分変換は、複数の前記配列について、その配列要素のサイズが異なるときには、前記サイズ比に応じて前記繰り返し回数がy回のループを案分する

ことを特徴とする請求項11に記載のプログラム変換装置。

[13] 前記案分変換は、複数の前記配列について、ループの繰り返し処理を1回を行うのに進むアドレスであるストライドが異なるときに、前記ストライド比に応じて前記繰り返し回数がy回のループを案分する

ことを特徴とする請求項11記載のプログラム変換装置。

[14] 前記案分変換は、内側ループを変換する際に、配分された各ループに対応する条件文を生成して、配分された各ループを同一の内側ループで実行されるように案分変換を行う

ことを特徴とする請求項11に記載のプログラム変換装置。

[15] 前記ループ構造変換手段は、

前記繰り返し回数がx回のループを、前記繰り返し回数がy回のループに分割する際に、x／yを演算した際の余りzが0でなければ、z回の繰り返し処理に対してピーリング処理を行ない、二重ループ変換を行う

ことを特徴とする請求項10に記載のプログラム変換装置。

[16] 前記ループ構造変換手段は、

前記余りzが0でなければ、内側ループのループ回数がy回であるかz回であるかを判定する判定する条件文を生成し、二重ループ変換を行う

ことを特徴とする請求項15記載のプログラム変換装置。

[17] 前記ループ構造変換手段は、ループの実行回数が不定な場合は、前記ループの実行回数を実行時に判定し、判定結果に基づいて繰り返し回数を動的に変化させるような二重ループ変換を行う

ことを特徴とする請求項10に記載のプログラム変換装置

[18] さらに、配列がキャッシュのラインサイズにアラインされているという情報を受け取る受け取り手段を備え、

前記命令配置手段は、前記繰り返し回数がx回のループに対して、当該ループにおけるx回の繰り返し処理で参照されるデータよりも一つ先のキャッシュのラインに記憶されるデータをプリフェッチするプリフェッチ命令を配置する

ことを特徴とする請求項10に記載のプログラム変換装置

[19] 前記最適化指定情報受け取り手段は、配列がキャッシュのラインのどの相対位置からアクセスを開始するかという情報を受け、

前記ループ構造変換手段は、当該情報に基づいて前記に二重ループ変換を行う

ことを特徴とする請求項10に記載のプログラム変換装置。

[20] 前記命令配置手段は、前記配列がキャッシュのラインサイズにアラインされていない場合には、前記繰り返し回数がx回のループに対して、当該ループにおけるx回の繰り返し処理で参照されるデータよりも二つ先のキャッシュのラインに記憶されるデータをプリフェッチするプリフェッチ命令を配置する

ことを特徴とする請求項10に記載のプログラム変換装置。

[21] 前記ループ構造変換手段は、前記配列がキャッシュのラインサイズにアラインされていない場合には、前記配列がキャッシュのラインのどの相対位置からアクセスを開始するかを判定し、判定結果に応じて二重ループ構造変換を行う

ことを特徴とする請求項10に記載のプログラム変換装置。

[22] さらに、着目する配列に関する情報を受け取る受け取り手段を備え、

前記ループ構造変換手段は、当該配列に対してのみ着目し、二重ループ変換を行う

ことを特徴とする請求項10に記載のプログラム変換装置。

[23] 前記ループ構造変換手段は、最内ループを1つのかたまりとみなして、外側のループに対してさらに二重ループ変換を行う

ことを特徴とする請求項1に記載のプログラム変換装置。

[24] 実行時に外部からの所定の応答を待つ命令を含む命令セットを備えたプロセッサ向けのプログラム変換方法であって、

入力プログラムに含まれる繰返し回数がx回のループを、繰返し回数がy回であるループを内側ループとし、繰返し回数がx／y回のループを外側ループとするネスト構造への変換である二重ループ変換を行うステップと、

前記内側ループの外部の位置に、前記命令を配置し、当該命令を含む出力プログラムに変換するステップと

を含むことを特徴とするプログラム変換方法。

[25] 実行時に外部からの所定の応答を待つ命令を含む命令セットを備えたプロセッサ向けのプログラム変換方法のプログラムであって、

入力プログラムに含まれる繰返し回数がx回のループを、繰返し回数がy回であるループを内側ループとし、繰返し回数がx／y回のループを外側ループとするネスト構造への変換である二重ループ変換を行うステップと、

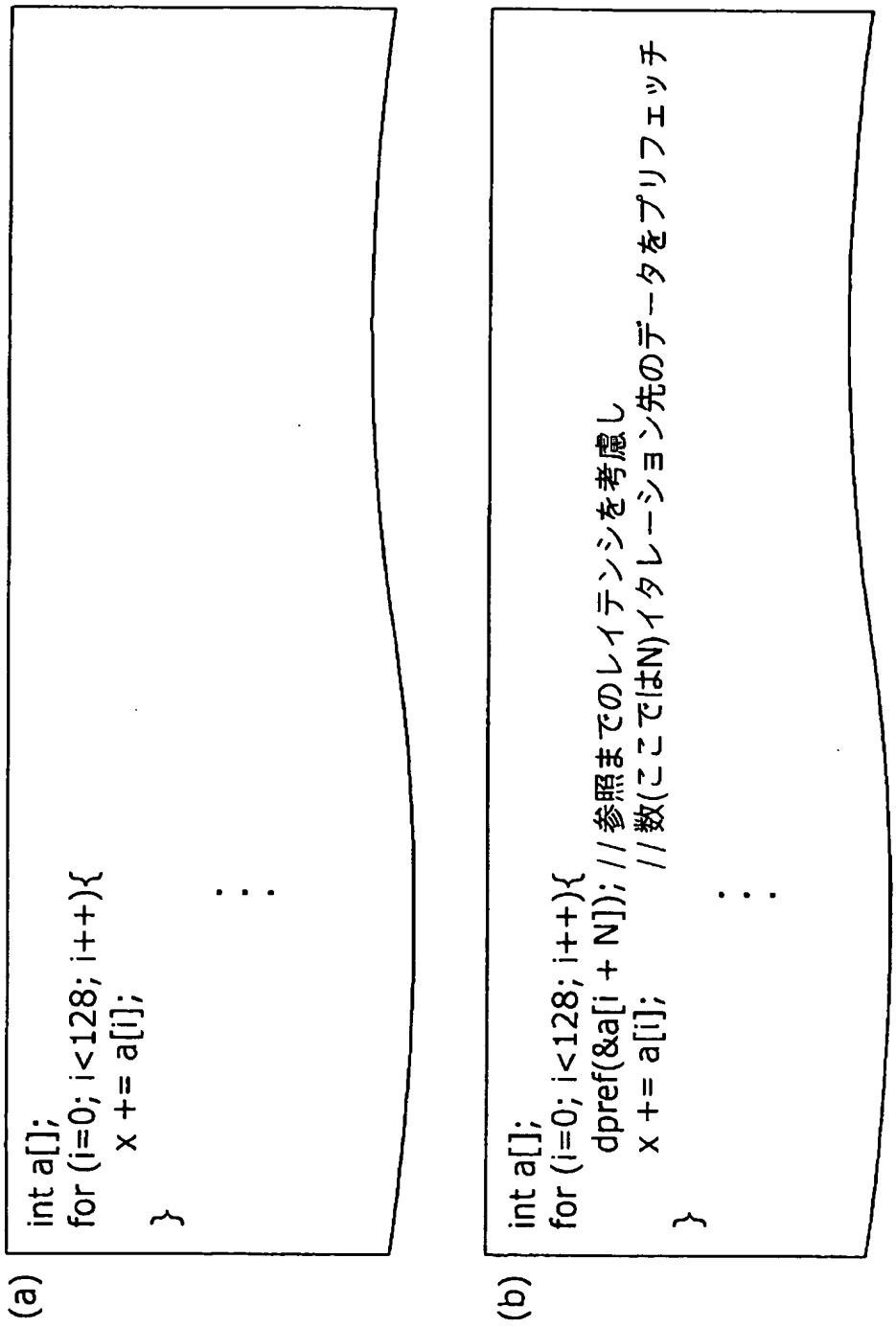
前記内側ループの外部の位置に、前記命令を配置し、当該命令を含む出力プログラムに変換するステップと

をコンピュータに実行させることを特徴とするプログラム。

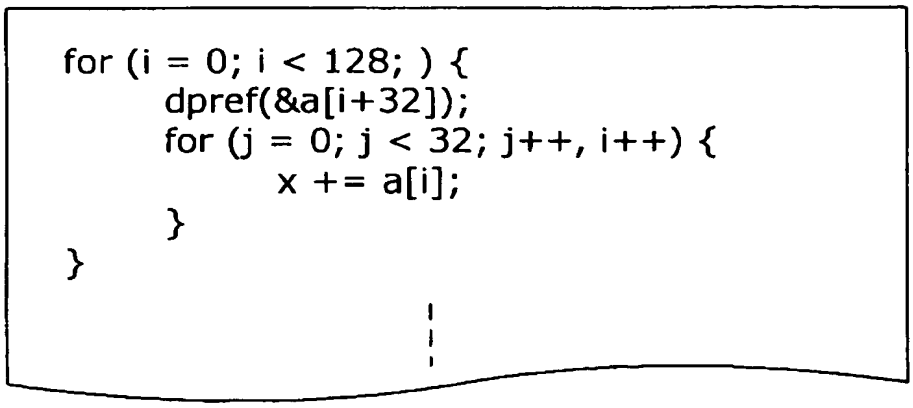
要 約 書

インターロックを引き起こす可能性のある命令を無駄に発行せずに、プログラム実行時の処理速度を向上させるコンパイラは、実行時にインターロックを起こす可能性のある命令を備えたプロセッサ向けのコンパイラであって、入力プログラムに対し、ループ回数がx回のループをループ回数がy回のループに分割し、前記ループ回数がy回のループを内側ループとし、ループ回数がx／y回のループを外側ループとする二重ループ変換を行うループ構造変換部(186)と、前記二重ループ変換後のプログラムに対して、インターロックを起こす可能性のある命令の配置を行う命令最適配置部(187)としてコンピュータを機能させることを特徴とする。

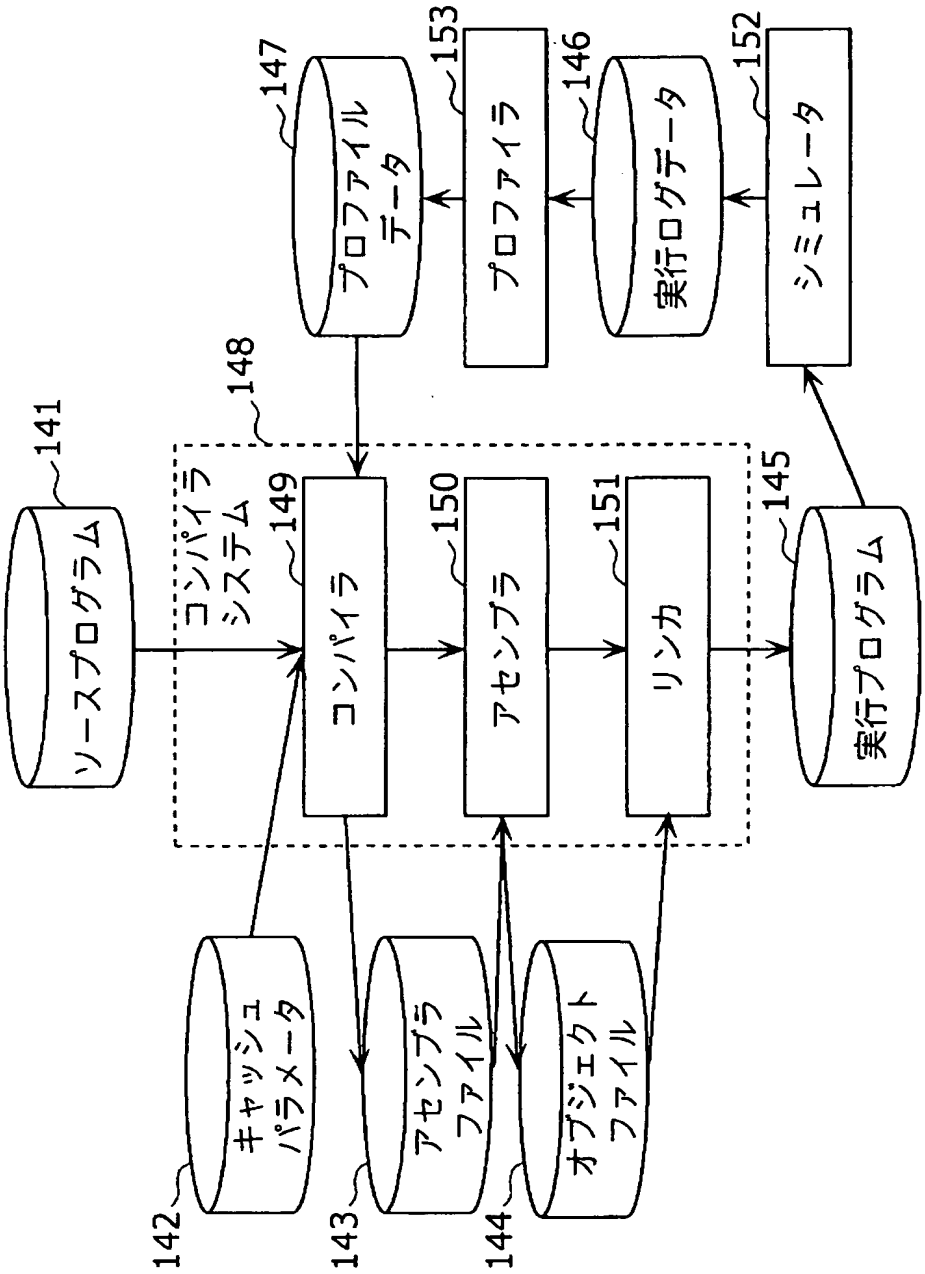
[図1]


(a)
int a[];
for (i=0; i<128; i++){
x += a[i];
}
. . .
(b)
int a[];
for (i=0; i<128; i++){
dpref(&a[i + N]); //参照までのレイテンシを考慮し
//数(ここではN)イタレーション先のデータをプリフェッチ
x += a[i];
}
. . .

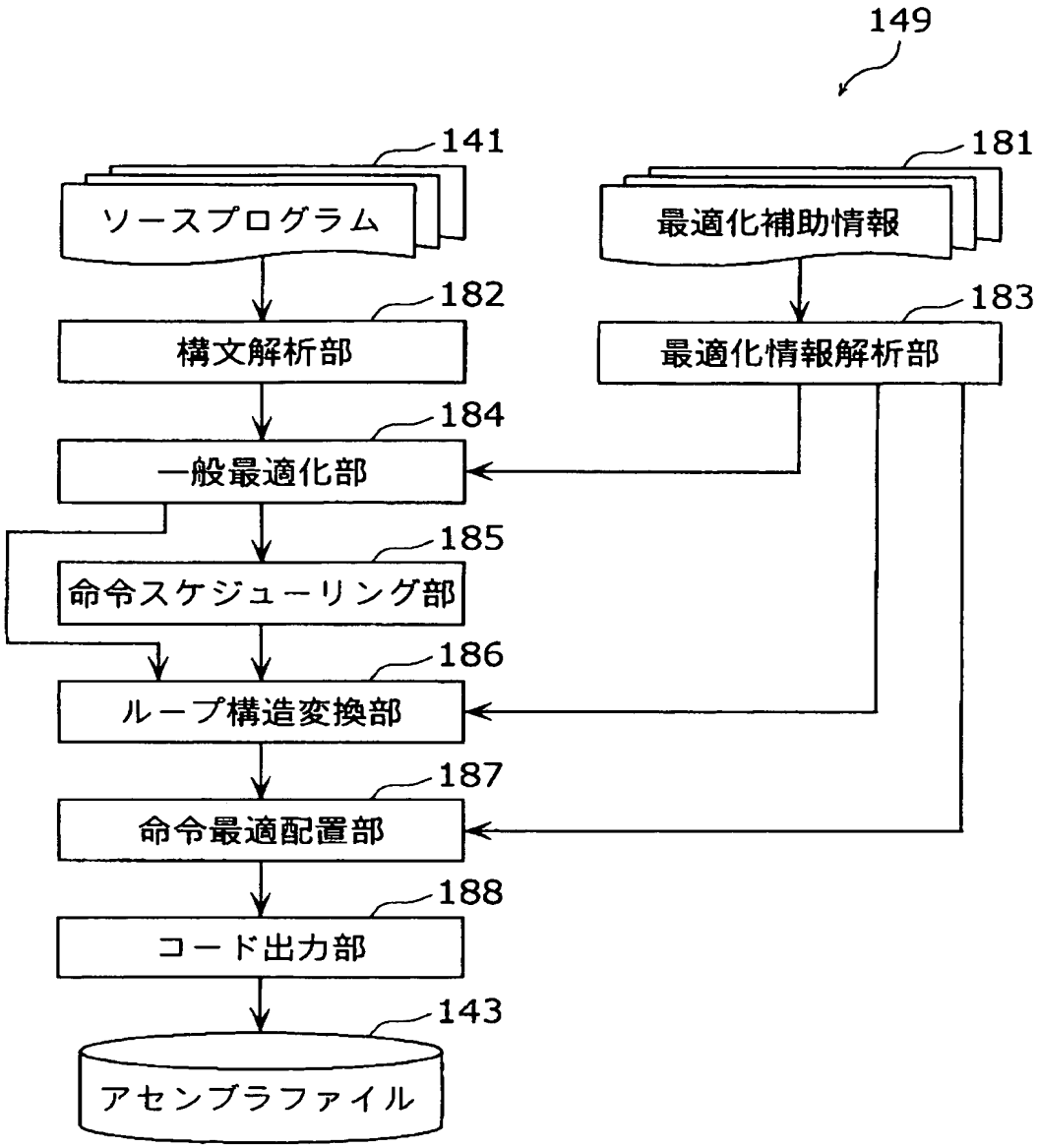
[図2]



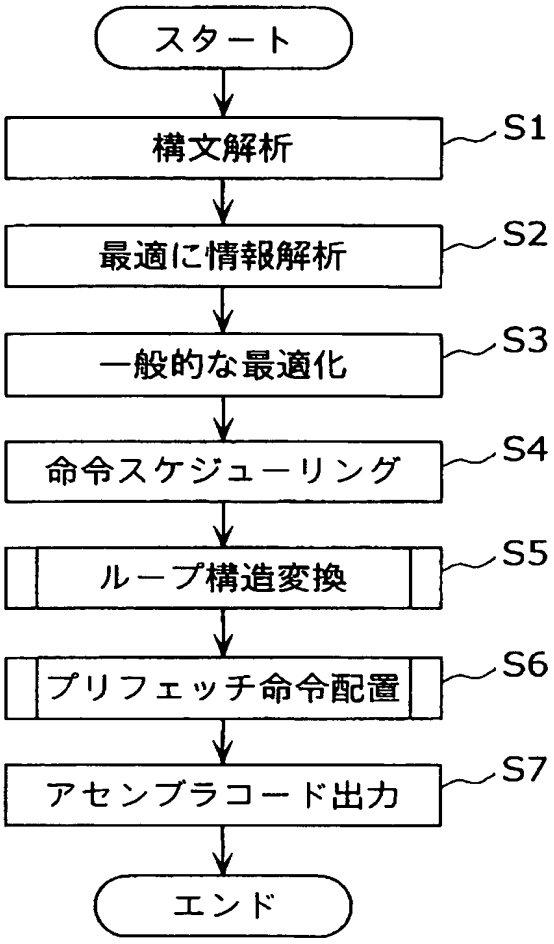
[図3]


ソースプログラム
141
コンパイラ
システム
148
コンパイラ
149
アセンブラ
150
リンカ
151
実行プログラム
145
プロファイル
データ
147
プロファイラ
153
実行ログデータ
146
シミュレータ
152
キャッシュ
パラメータ
142
アセンブラ
ファイル
143
オブジェクト
ファイル
144

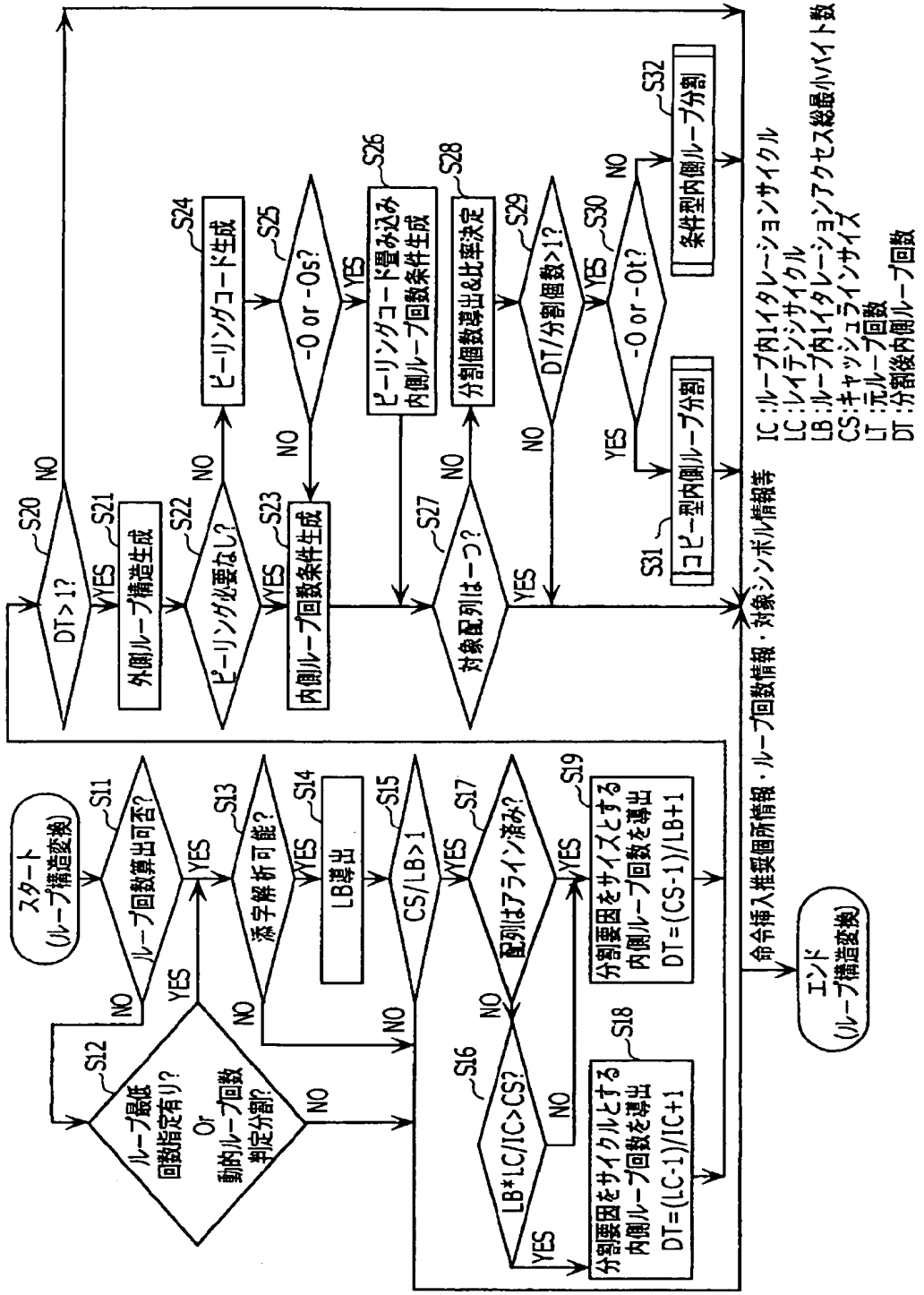
[図4]


149
141
ソースプログラム
181
最適化補助情報
182
構文解析部
183
最適化情報解析部
184
一般最適化部
185
命令スケジューリング部
186
ループ構造変換部
187
命令最適配置部
188
コード出力部
143
アセンブラファイル

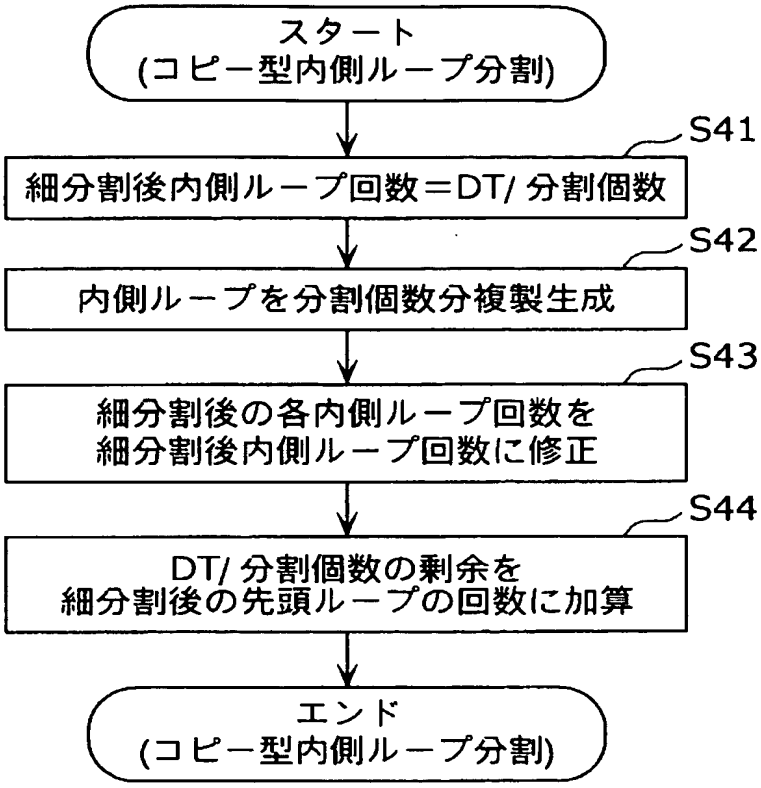
[図5]


スタート
構文解析
S1
最適に情報解析
S2
一般的な最適化
S3
命令スケジューリング
S4
ループ構造変換
S5
プリフェッチ命令配置
S6
アセンブラコード出力
S7
エンド

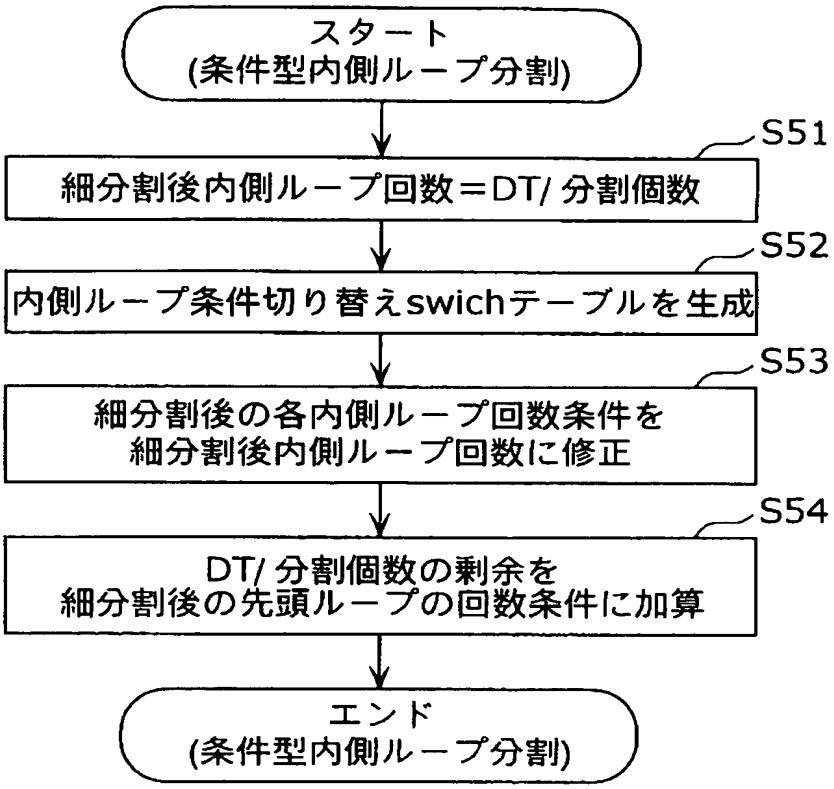
[図6]


スタート
(ループ構造変換)
S11
ループ回数算出可否?
YES
NO
S12
ループ最低
回数指定有り?
Or
動的ループ回数
判定分割?
YES
NO
S13
添字解析可能?
YES
NO
S14
LB導出
S15
CS/LB>1
YES
NO
S16
LB*LC/IC>CS?
YES
NO
S17
配列はアライン済み?
YES
NO
S18
分割要因をサイクルとする
内側ループ回数を導出
DT=(LC-1)/IC+1
S19
分割要因をサイズとする
内側ループ回数を導出
DT=(CS-1)/LB+1
S20
DT>1?
YES
NO
S21
外側ループ構造生成
S22
ピーリング必要なし?
YES
NO
S23
内側ループ回数条件生成
S24
ピーリングコード生成
S25
-O or -Os?
YES
NO
S26
ピーリングコード置み込み
内側ループ回数条件生成
S27
対象配列は一つ?
YES
NO
S28
分割個数導出&比率決定
S29
DT/分割個数>1?
YES
NO
S30
-O or -Ot?
YES
NO
S31
コピー型内側ループ分割
S32
条件型内側ループ分割
命令挿入推奨個所情報・ループ回数情報・対象シンボル情報等
エンド
(ループ構造変換)
IC :ループ内1イタレーションサイクル
LC :レイテンシサイクル
LB :ループ内1イタレーションアクセス総最小バイト数
CS:キャッシュラインサイズ
LT :元ループ回数
DT :分割後内側ループ回数

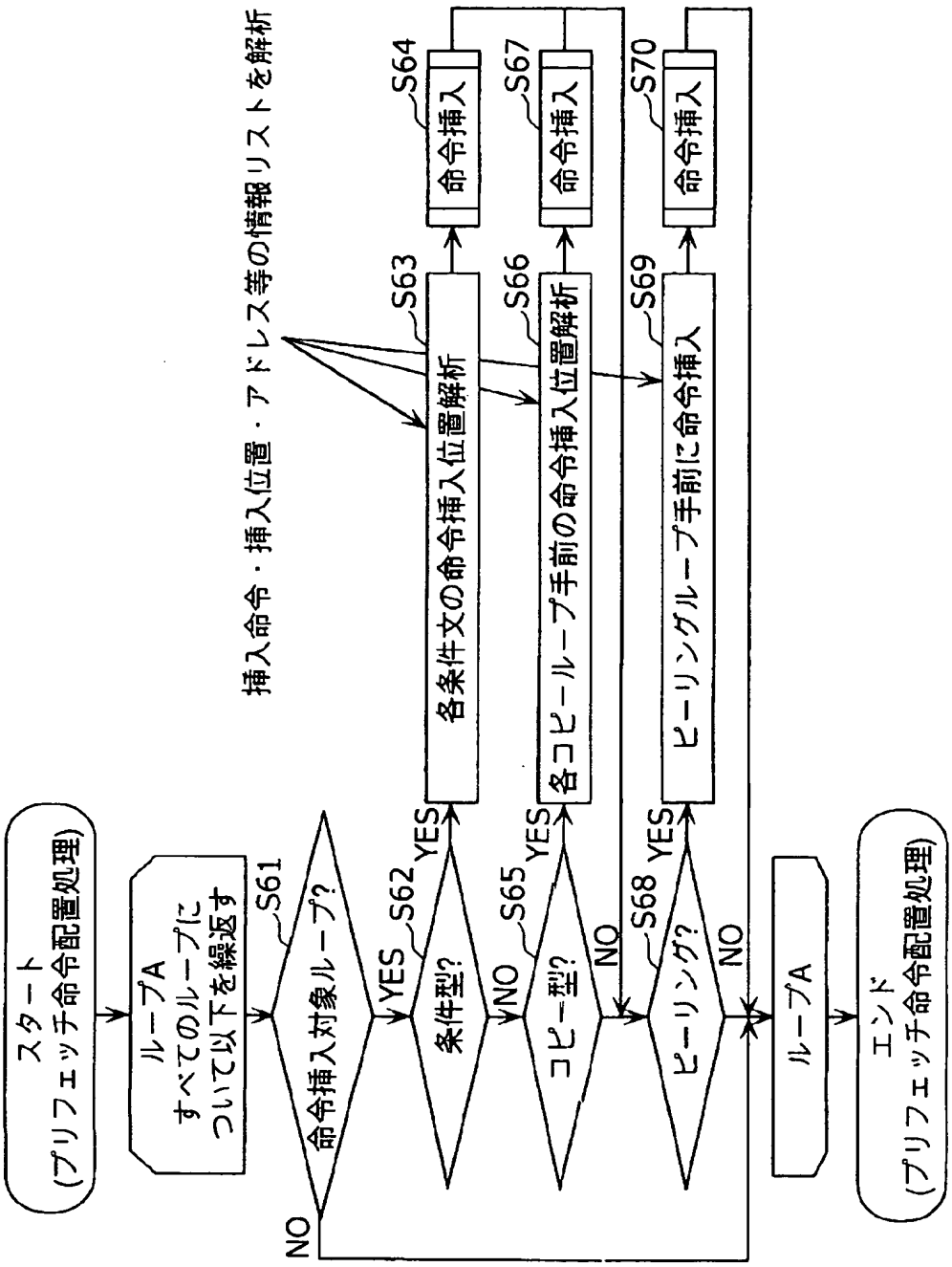
[図7]


スタート
(コピー型内側ループ分割)
S41
細分割後内側ループ回数＝DT/ 分割個数
S42
内側ループを分割個数分複製生成
S43
細分割後の各内側ループ回数を
細分割後内側ループ回数に修正
S44
DT/ 分割個数の剰余を
細分割後の先頭ループの回数に加算
エンド
(コピー型内側ループ分割)

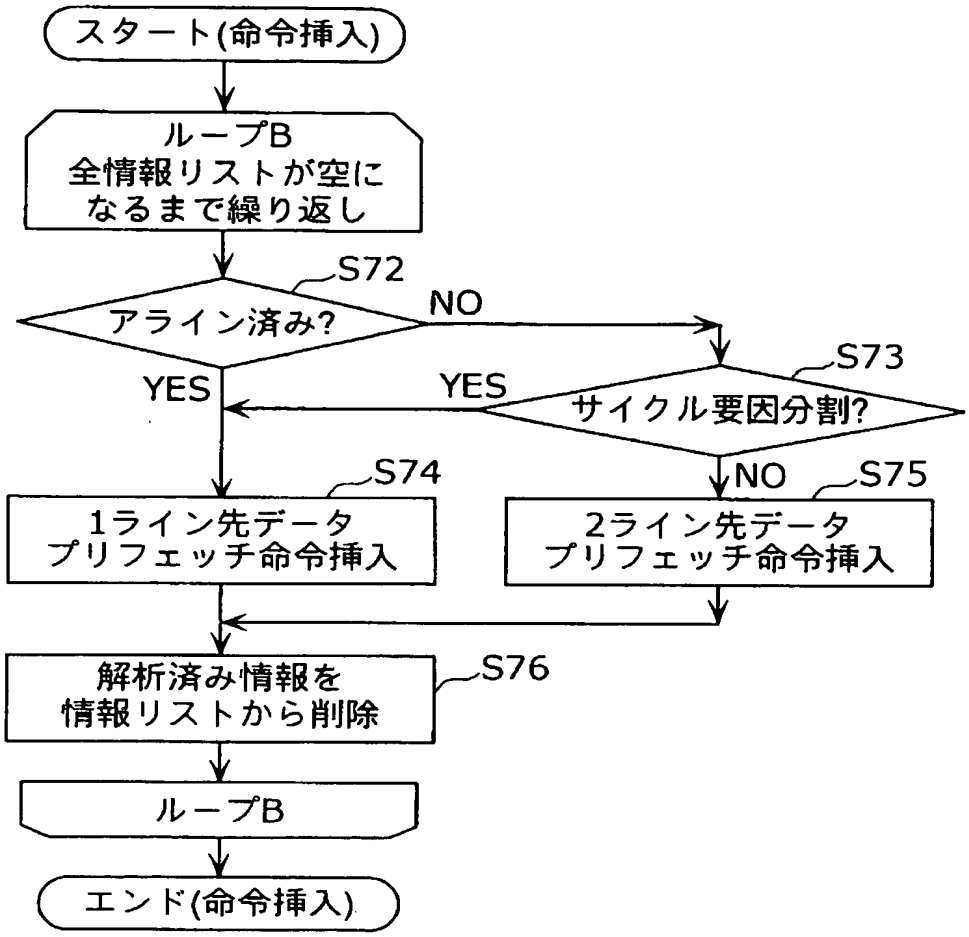
[図8]


スタート
(条件型内側ループ分割)
S51
細分割後内側ループ回数＝DT/ 分割個数
S52
内側ループ条件切り替えswichテーブルを生成
S53
細分割後の各内側ループ回数条件を
細分割後内側ループ回数に修正
S54
DT/ 分割個数の剰余を
細分割後の先頭ループの回数条件に加算
エンド
(条件型内側ループ分割)

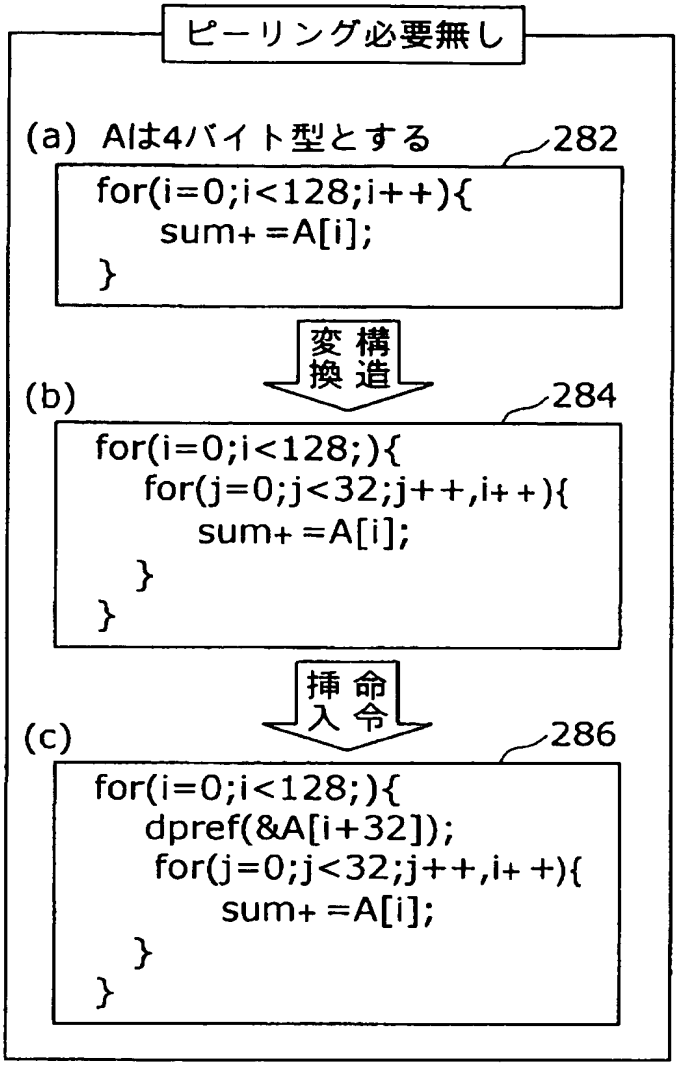
[図9]


スタート
(プリフェッチ命令配置処理)
ループA
すべてのループについて以下を繰返す
S61
命令挿入対象ループ?
YES
NO
S62
条件型?
YES
NO
S63
各条件文の命令挿入位置解析
S64
命令挿入
S65
コピー型?
YES
NO
S66
各コピーループ手前の命令挿入位置解析
S67
命令挿入
S68
ピーリング?
YES
NO
S69
ピーリングループ手前に命令挿入
S70
命令挿入
挿入命令・挿入位置・アドレス等の情報リストを解析
ループA
エンド
(プリフェッチ命令配置処理)

[図10]


スタート(命令挿入)
ループB
全情報リストが空に
なるまで繰り返し
S72
アライン済み?
NO
YES
S73
YES
サイクル要因分割?
NO
S74
1ライン先データ
プリフェッチ命令挿入
S75
2ライン先データ
プリフェッチ命令挿入
解析済み情報を
情報リストから削除
S76
ループB
エンド(命令挿入)

[図11]


ピーリング必要無し
(a) Aは4バイト型とする
282
for(i=0;i<128;i++){
sum+=A[i];
}
変換
構造
(b)
284
for(i=0;i<128;){
for(j=0;j<32;j++,i++){
sum+=A[i];
}
}
挿入
命令
(c)
286
for(i=0;i<128;){
dpref(&A[i+32]);
for(j=0;j<32;j++,i++){
sum+=A[i];
}
}

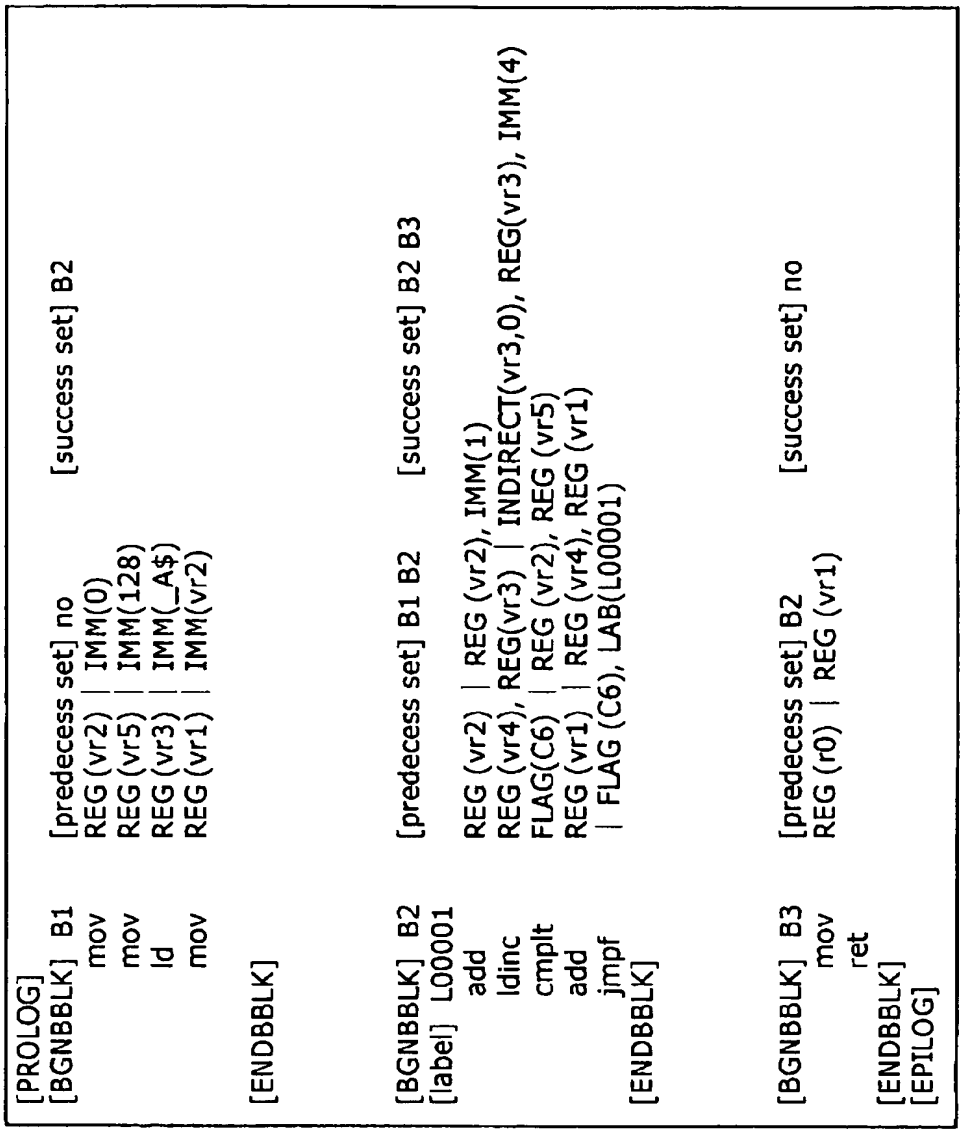
[図12]

入力C言語プログラムソース　240

```
int A[1000];

int main(void)
{
        int i:
        int sum = 0;

        for ( i=0; j<128;i++) {
              sum += A[ i ];
        }

        return sum;
}
```

[図13]

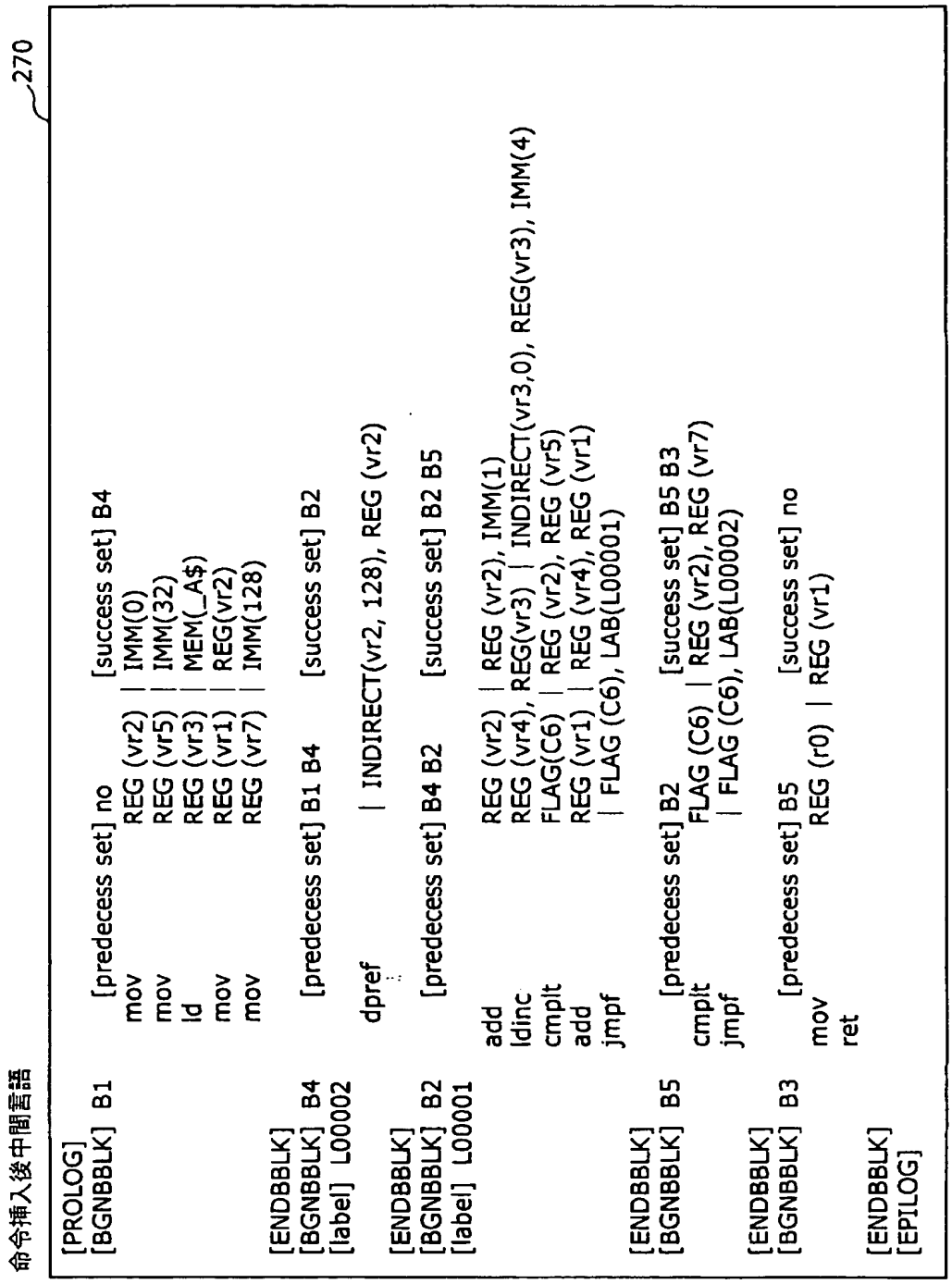
変換部入力中間言語



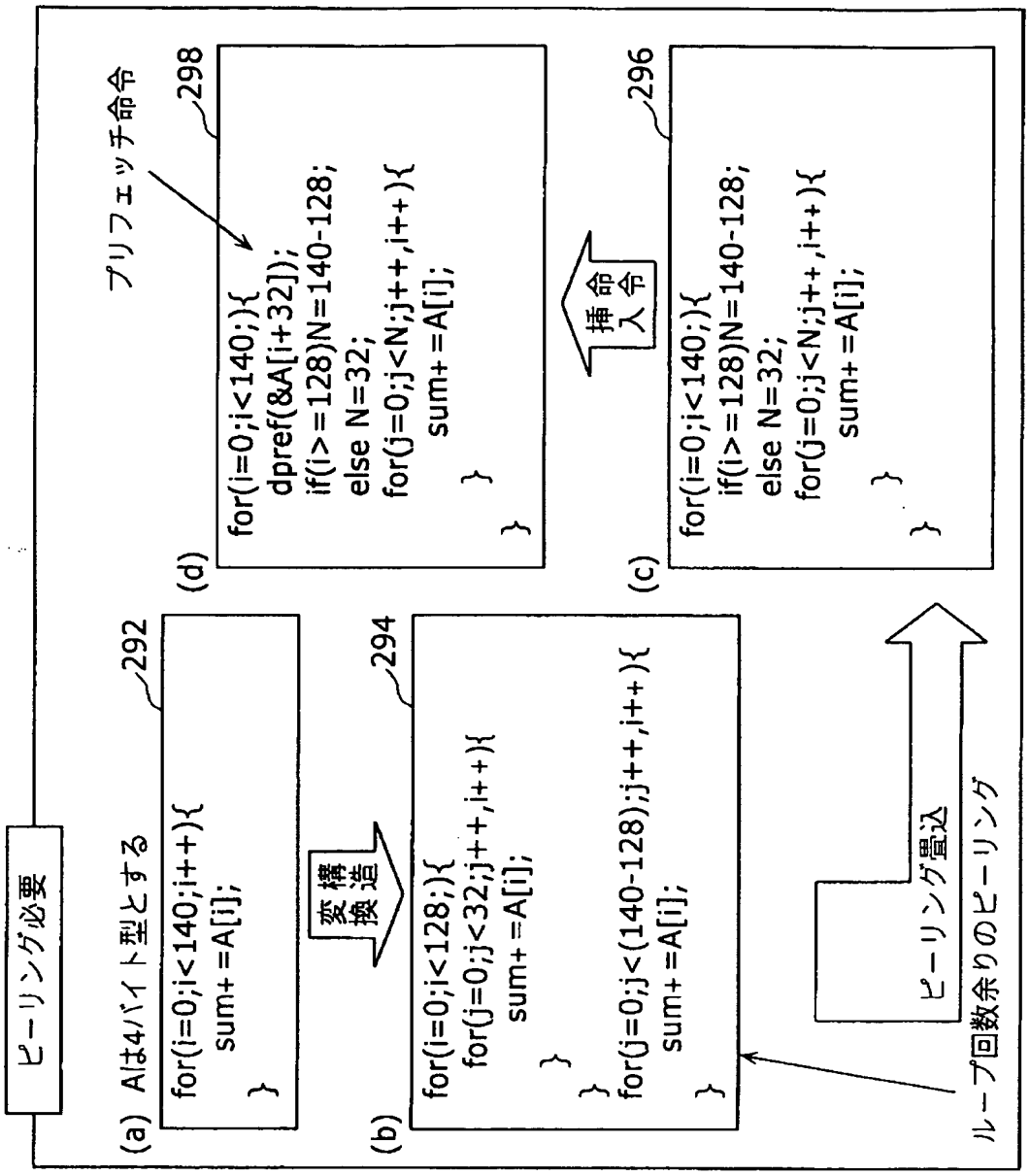
変換後中間言語

```
[PROLOG]
[BGNBBLK] B1        [predecess set] no          [success set] B4
                mov             REG (vr2) | IMM(0)
                mov             REG (vr5) | IMM(32)
                ld              REG (vr3) | MEM(_A$)
                mov             REG (vr1) | REG(vr2)
                mov             REG (vr7) | IMM(128)
[ENDBBLK]
[BGNBBLK] B4        [predecess set] B1 B4       [success set] B2
[label] L00002

[ENDBBLK]
[BGNBBLK] B2        [predecess set] B4 B2       [success set] B2 B5
[label] L00001
                add             REG (vr2) | REG (vr2), IMM(1)
                ldinc           REG (vr4), REG(vr3) | INDIRECT(vr3,0), REG(vr3), IMM(4)
                cmplt           FLAG(C6) | REG (vr2), REG (vr5)
                add             REG (vr1) | REG (vr4), REG (vr1)
                jmpf             | FLAG (C6), LAB(L00001)
[ENDBBLK]
[BGNBBLK] B5        [predecess set] B2          [success set] B5 B3
                cmplt           FLAG (C6) | REG (vr2), REG (vr7)
                jmpf             | FLAG (C6), LAB(L00002)
[ENDBBLK]
[BGNBBLK] B3        [predecess set] B5          [success set] no
                mov             REG (r0) | REG (vr1)
                ret
[ENDBBLK]
[EPILOG]
```

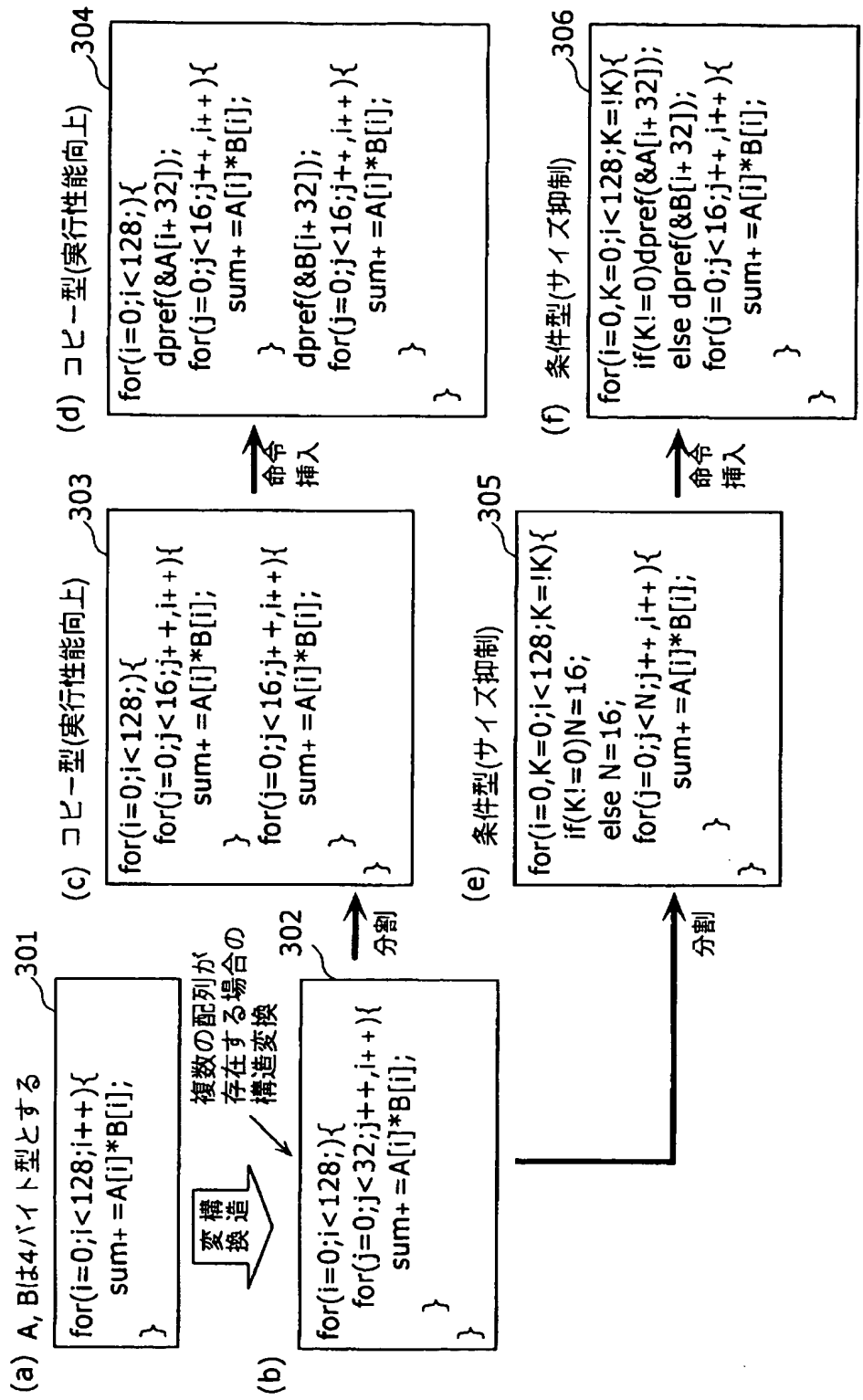
[図15]


命令挿入後中間言語
270
[PROLOG]
[BGNBBLK] B1 [predecess set] no [success set] B4
mov REG (vr2) | IMM(0)
mov REG (vr5) | IMM(32)
ld REG (vr3) | MEM(_A$)
mov REG (vr1) | REG(vr2)
mov REG (vr7) | IMM(128)
[ENDBBLK]
[BGNBBLK] B4 [predecess set] B1 B4 [success set] B2
[label] L00002
dpref | INDIRECT(vr2, 128), REG (vr2)
[ENDBBLK]
[BGNBBLK] B2 [predecess set] B4 B2 [success set] B2 B5
[label] L00001
add REG (vr2) | REG (vr2), IMM(1)
ldinc REG (vr4), REG(vr3) | INDIRECT(vr3,0), REG(vr3), IMM(4)
cmplt FLAG(C6) | REG (vr2), REG (vr5)
add REG (vr1) | REG (vr4), REG (vr1)
jmpf | FLAG (C6), LAB(L00001)
[ENDBBLK]
[BGNBBLK] B5 [predecess set] B2 [success set] B5 B3
cmplt FLAG (C6) | REG (vr2), REG (vr7)
jmpf | FLAG (C6), LAB(L00002)
[ENDBBLK]
[BGNBBLK] B3 [predecess set] B5 [success set] no
mov REG (r0) | REG (vr1)
ret
[ENDBBLK]
[EPILOG]

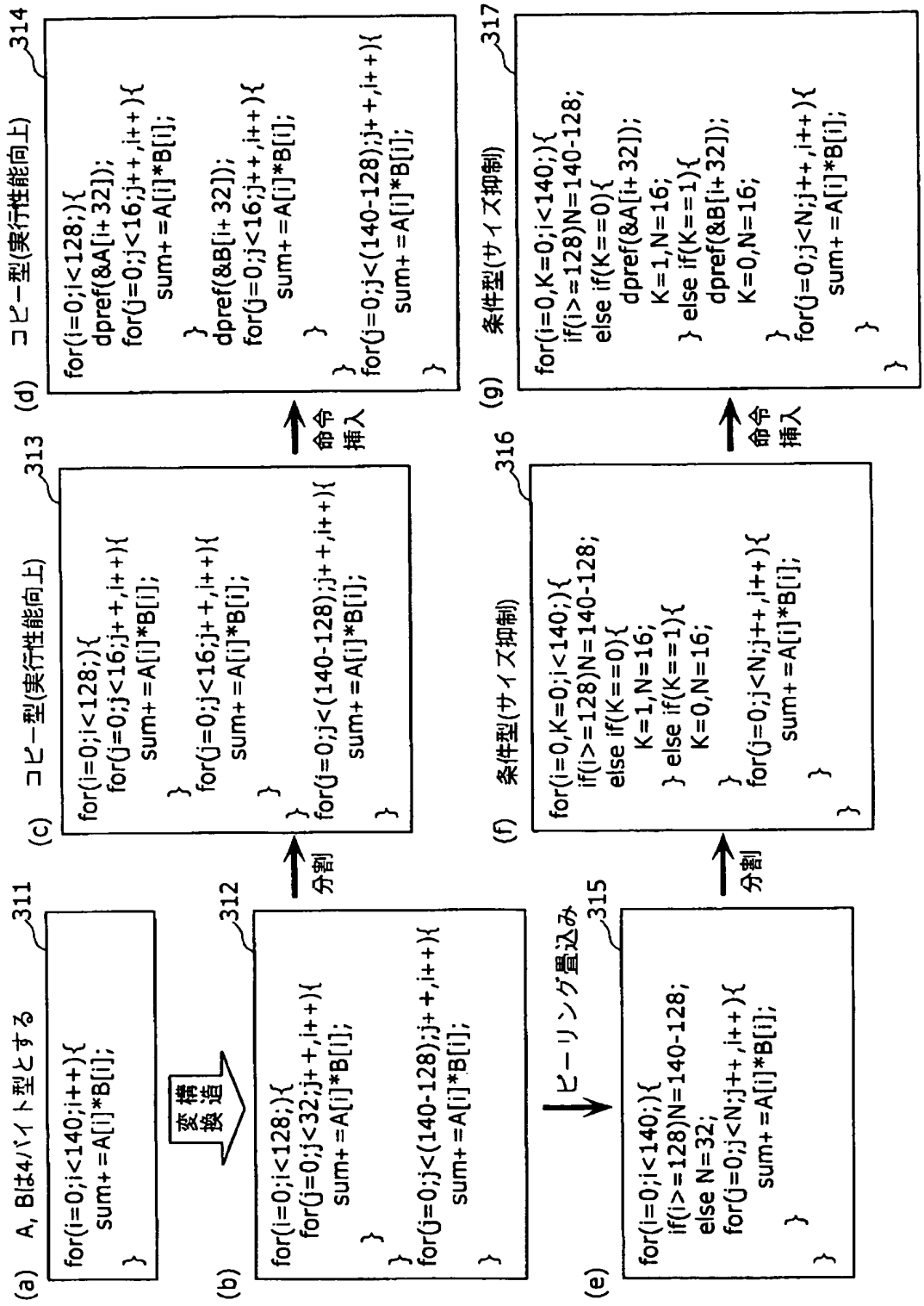
[図16]


ヒーリング必要
(a) Aは4バイト型とする
292
for(i=0;i<140;i++){
sum+=A[i];
}
変構換造
(b)
294
for(i=0;i<128;){
for(j=0;j<32;j++,i++){
sum+=A[i];
}
}
for(j=0;j<(140-128);j++,i++){
sum+=A[i];
}
ループ回数余りのヒーリング
ヒーリング置込
(c)
296
for(i=0;i<140;){
if(i>=128)N=140-128;
else N=32;
for(j=0;j<N;j++,i++){
sum+=A[i];
}
}
挿命入令
(d)
298
for(i=0;i<140;){
dpref(&A[i+32]);
if(i>=128)N=140-128;
else N=32;
for(j=0;j<N;j++,i++){
sum+=A[i];
}
}
プリフェッチ命令

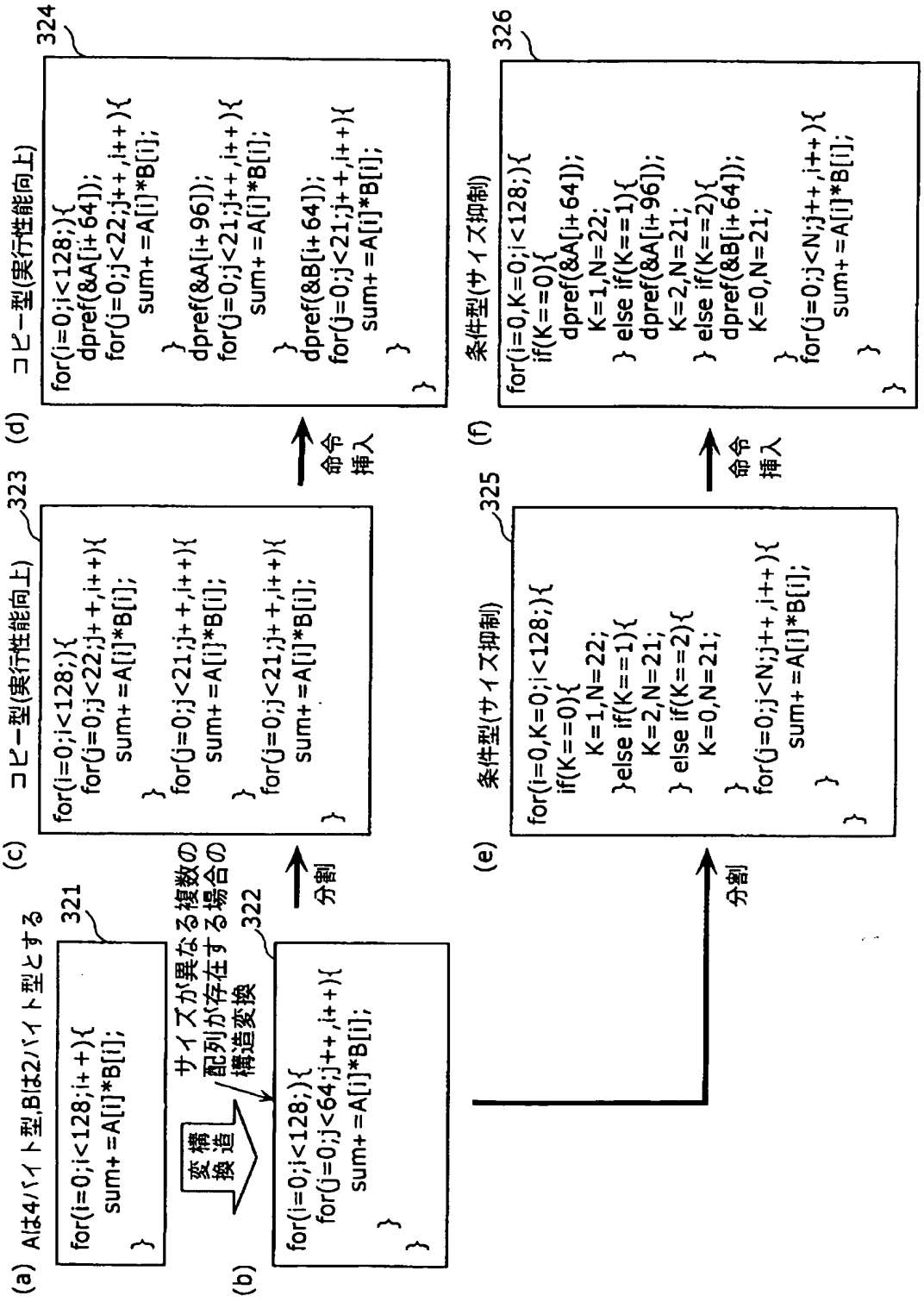
[図17]


(a) A, Bは4バイト型とする
301
for(i=0;i<128;i++){
sum+ =A[i]*B[i];
}
変換
構造
複数の配列が
存在する場合の
構造変換
(b)
302
for(i=0;i<128;){
for(j=0;j<32;j++,i++){
sum+ =A[i]*B[i];
}
}
分割
(c) コピー型(実行性能向上)
303
for(i=0;i<128;){
for(j=0;j<16;j++,i++){
sum+ =A[i]*B[i];
}
for(j=0;j<16;j++,i++){
sum+ =A[i]*B[i];
}
}
命令
挿入
(d) コピー型(実行性能向上)
304
for(i=0;i<128;){
dpref(&A[i+32]);
for(j=0;j<16;j++,i++){
sum+ =A[i]*B[i];
}
dpref(&B[i+32]);
for(j=0;j<16;j++,i++){
sum+ =A[i]*B[i];
}
}
(e) 条件型(サイズ抑制)
305
for(i=0,K=0;i<128;K=!K){
if(K!=0)N=16;
else N=16;
for(j=0;j<N;j++,i++){
sum+ =A[i]*B[i];
}
}
分割
(f) 条件型(サイズ抑制)
306
for(i=0,K=0;i<128;K=!K){
if(K!=0)dpref(&A[i+32]);
else dpref(&B[i+32]);
for(j=0;j<16;j++,i++){
sum+ =A[i]*B[i];
}
}
命令
挿入

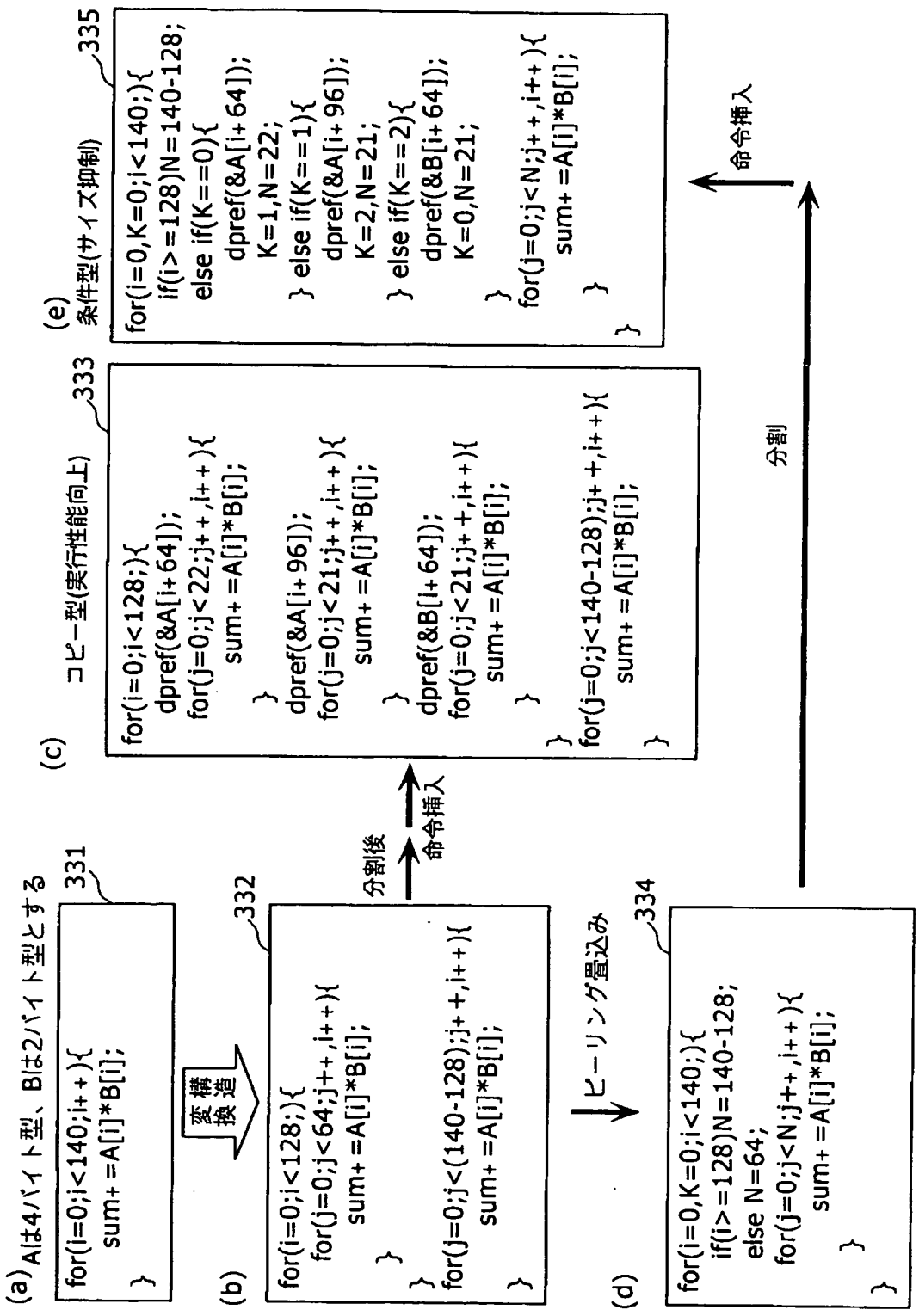
[図18]


(a) A, Bは4バイト型とする
311
for(i=0;i<140;i++){
sum+=A[i]*B[i];
}
変構換造
(b)
312
for(i=0;i<128;){
for(j=0;j<32;j++,i++){
sum+=A[i]*B[i];
}
}
for(j=0;j<(140-128);j++,i++){
sum+=A[i]*B[i];
}
分割
(c) コピー型(実行性能向上)
313
for(i=0;i<128;){
for(j=0;j<16;j++,i++){
sum+=A[i]*B[i];
}
for(j=0;j<16;j++,i++){
sum+=A[i]*B[i];
}
}
for(j=0;j<(140-128);j++,i++){
sum+=A[i]*B[i];
}
命令挿入
(d) コピー型(実行性能向上)
314
for(i=0;i<128;){
dpref(&A[i+32]);
for(j=0;j<16;j++,i++){
sum+=A[i]*B[i];
}
dpref(&B[i+32]);
for(j=0;j<16;j++,i++){
sum+=A[i]*B[i];
}
}
for(j=0;j<(140-128);j++,i++){
sum+=A[i]*B[i];
}
ヒーリング置込み
(e)
315
for(i=0;i<140;){
if(i>=128)N=140-128;
else N=32;
for(j=0;j<N;j++,i++){
sum+=A[i]*B[i];
}
}
分割
(f) 条件型(サイズ抑制)
316
for(i=0,K=0;i<140;){
if(i>=128)N=140-128;
else if(K==0){
K=1,N=16;
} else if(K==1){
K=0,N=16;
}
for(j=0;j<N;j++,i++){
sum+=A[i]*B[i];
}
}
命令挿入
(g) 条件型(サイズ抑制)
317
for(i=0,K=0;i<140;){
if(i>=128)N=140-128;
else if(K==0){
dpref(&A[i+32]);
K=1,N=16;
} else if(K==1){
dpref(&B[i+32]);
K=0,N=16;
}
for(j=0;j<N;j++,i++){
sum+=A[i]*B[i];
}
}

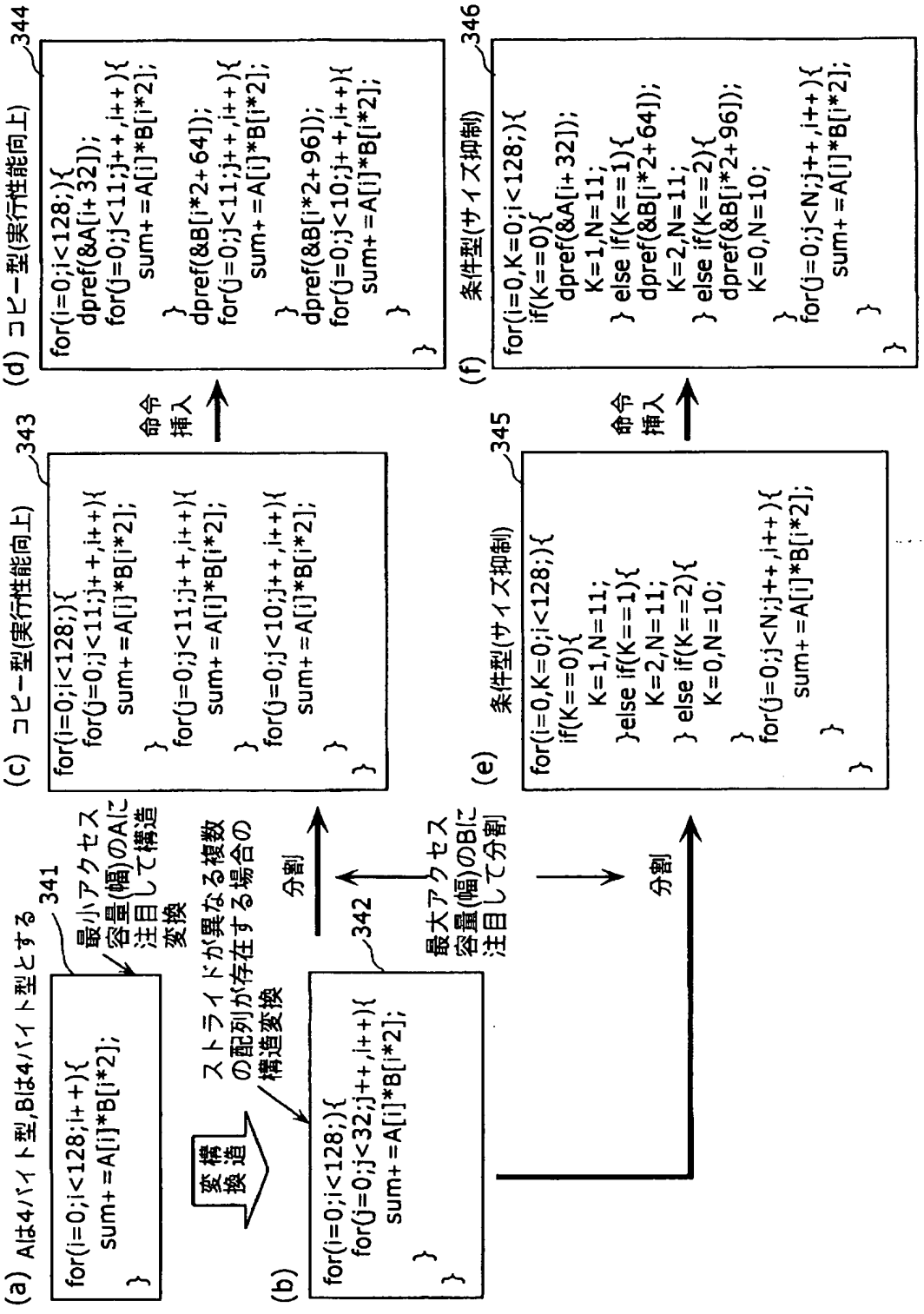
[図19]


(a) Aは4バイト型,Bは2バイト型とする
321
for(i=0;i<128;i++){
sum+=A[i]*B[i];
}
変構換造
サイズが異なる複数の配列が存在する場合の構造変換
(b)
322
for(i=0;i<128;){
for(j=0;j<64;j++,i++){
sum+=A[i]*B[i];
}
}
分割
(c) コピー型(実行性能向上)
323
for(i=0;i<128;){
for(j=0;j<22;j++,i++){
sum+=A[i]*B[i];
}
for(j=0;j<21;j++,i++){
sum+=A[i]*B[i];
}
for(j=0;j<21;j++,i++){
sum+=A[i]*B[i];
}
}
命令挿入
(d) コピー型(実行性能向上)
324
for(i=0;i<128;){
dpref(&A[i+64]);
for(j=0;j<22;j++,i++){
sum+=A[i]*B[i];
}
dpref(&A[i+96]);
for(j=0;j<21;j++,i++){
sum+=A[i]*B[i];
}
dpref(&B[i+64]);
for(j=0;j<21;j++,i++){
sum+=A[i]*B[i];
}
}
(e) 条件型(サイズ抑制)
325
for(i=0,K=0;i<128;){
if(K==0){
K=1,N=22;
}else if(K==1){
K=2,N=21;
} else if(K==2){
K=0,N=21;
}
for(j=0;j<N;j++,i++){
sum+=A[i]*B[i];
}
}
分割
(f) 条件型(サイズ抑制)
326
for(i=0,K=0;i<128;){
if(K==0){
dpref(&A[i+64]);
K=1,N=22;
} else if(K==1){
dpref(&A[i+96]);
K=2,N=21;
} else if(K==2){
dpref(&B[i+64]);
K=0,N=21;
}
for(j=0;j<N;j++,i++){
sum+=A[i]*B[i];
}
}
命令挿入

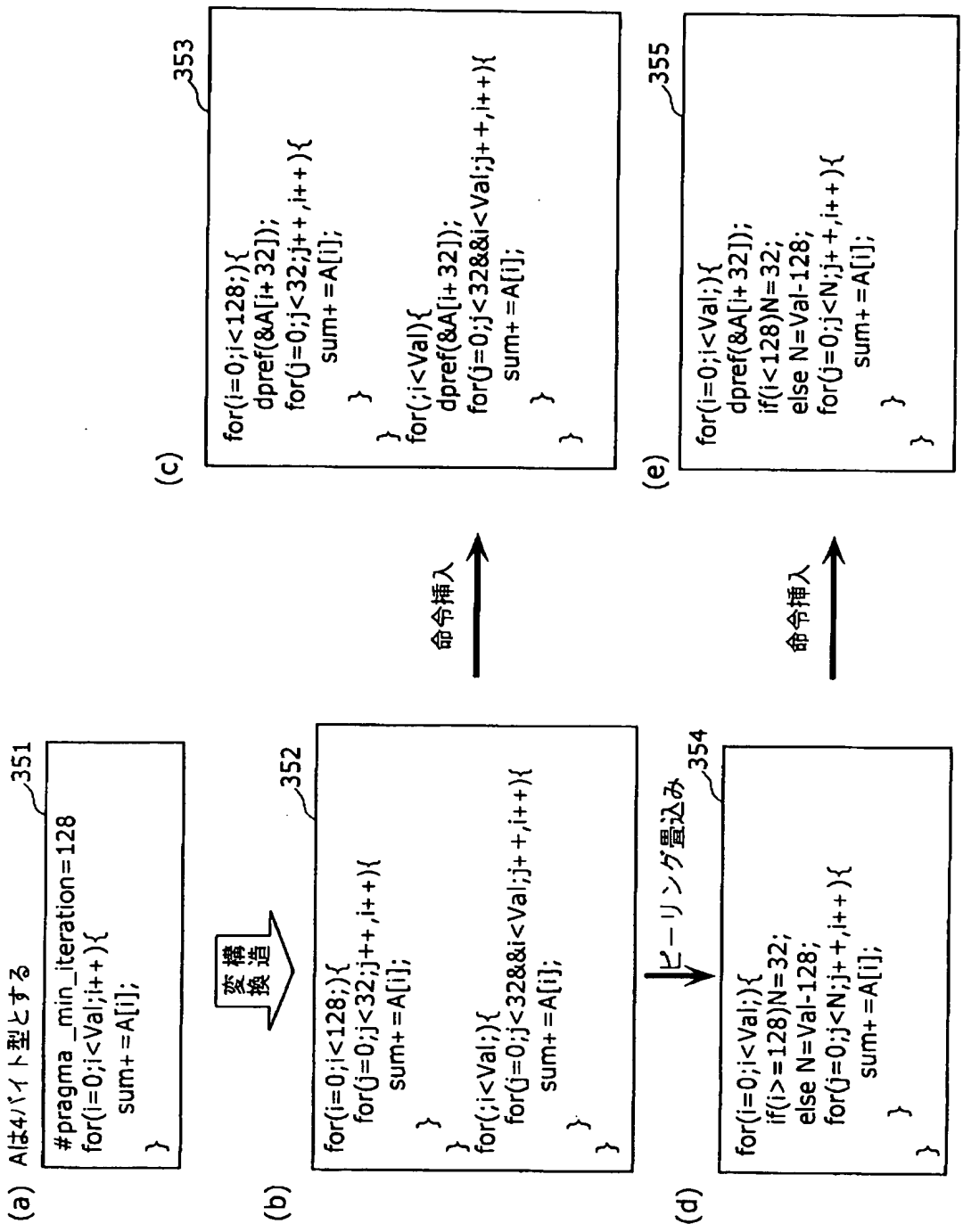
[図20]


(a) Aは4バイト型、Bは2バイト型とする
331
for(i=0;i<140;i++){
sum+=A[i]*B[i];
}
変構換造
(b)
332
for(i=0;i<128;){
for(j=0;j<64;j++,i++){
sum+=A[i]*B[i];
}
}
for(j=0;j<(140-128);j++,i++){
sum+=A[i]*B[i];
}
分割後
命令挿入
(c) コピー型(実行性能向上)
333
for(i=0;i<128;){
dpref(&A[i+64]);
for(j=0;j<22;j++,i++){
sum+=A[i]*B[i];
}
dpref(&A[i+96]);
for(j=0;j<21;j++,i++){
sum+=A[i]*B[i];
}
dpref(&B[i+64]);
for(j=0;j<21;j++,i++){
sum+=A[i]*B[i];
}
}
for(j=0;j<140-128);j++,i++){
sum+=A[i]*B[i];
}
(e) 条件型(サイズ抑制)
335
for(i=0,K=0;i<140;){
if(i>=128)N=140-128;
else if(K==0){
dpref(&A[i+64]);
K=1,N=22;
} else if(K==1){
dpref(&A[i+96]);
K=2,N=21;
} else if(K==2){
dpref(&B[i+64]);
K=0,N=21;
}
for(j=0;j<N;j++,i++){
sum+=A[i]*B[i];
}
}
ヒーリング置込み
(d)
334
for(i=0,K=0;i<140;){
if(i>=128)N=140-128;
else N=64;
for(j=0;j<N;j++,i++){
sum+=A[i]*B[i];
}
}
分割
命令挿入

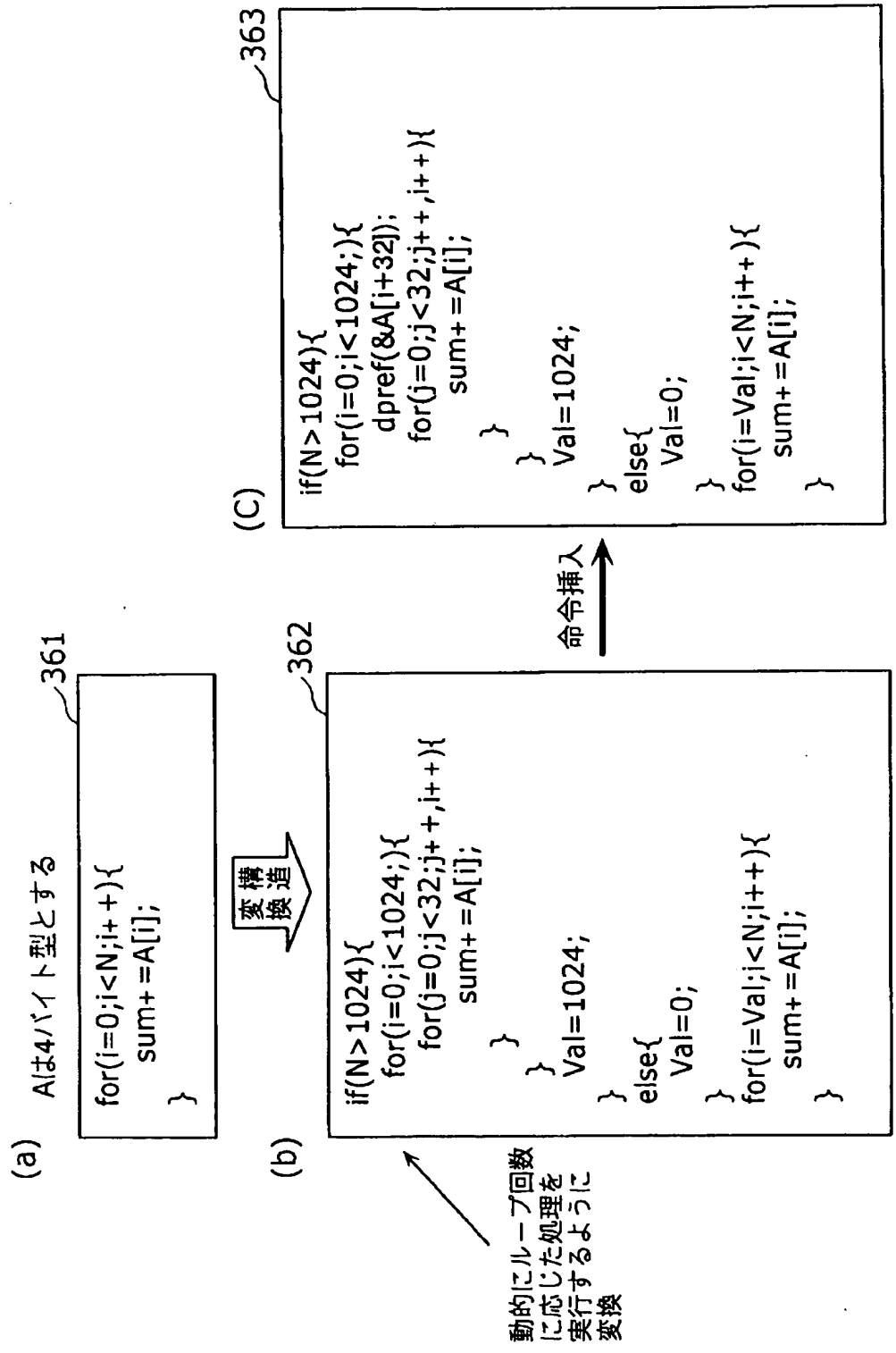
[図21]


(a) Aは4バイト型,Bは4バイト型とする
341
for(i=0;i<128;i++){
sum+=A[i]*B[i*2];
}
最小アクセス容量(幅)のAに注目して構造変換
変換構造
ストライドが異なる複数の配列が存在する場合の構造変換
(b)
342
for(i=0;i<128;){
for(j=0;j<32;j++,i++){
sum+=A[i]*B[i*2];
}
}
最大アクセス容量(幅)のBに注目して分割
分割
(c) コピー型(実行性能向上)
343
for(i=0;i<128;){
for(j=0;j<11;j++,i++){
sum+=A[i]*B[i*2];
}
for(j=0;j<11;j++,i++){
sum+=A[i]*B[i*2];
}
for(j=0;j<10;j++,i++){
sum+=A[i]*B[i*2];
}
}
命令挿入
(d) コピー型(実行性能向上)
344
for(i=0;i<128;){
dpref(&A[i+32]);
for(j=0;j<11;j++,i++){
sum+=A[i]*B[i*2];
}
dpref(&B[i*2+64]);
for(j=0;j<11;j++,i++){
sum+=A[i]*B[i*2];
}
dpref(&B[i*2+96]);
for(j=0;j<10;j++,i++){
sum+=A[i]*B[i*2];
}
}
(e) 条件型(サイズ抑制)
345
for(i=0,K=0;i<128;){
if(K==0){
K=1,N=11;
}else if(K==1){
K=2,N=11;
} else if(K==2){
K=0,N=10;
}
for(j=0;j<N;j++,i++){
sum+=A[i]*B[i*2];
}
}
分割
命令挿入
(f) 条件型(サイズ抑制)
346
for(i=0,K=0;i<128;){
if(K==0){
dpref(&A[i+32]);
K=1,N=11;
} else if(K==1){
dpref(&B[i*2+64]);
K=2,N=11;
} else if(K==2){
dpref(&B[i*2+96]);
K=0,N=10;
}
for(j=0;j<N;j++,i++){
sum+=A[i]*B[i*2];
}
}

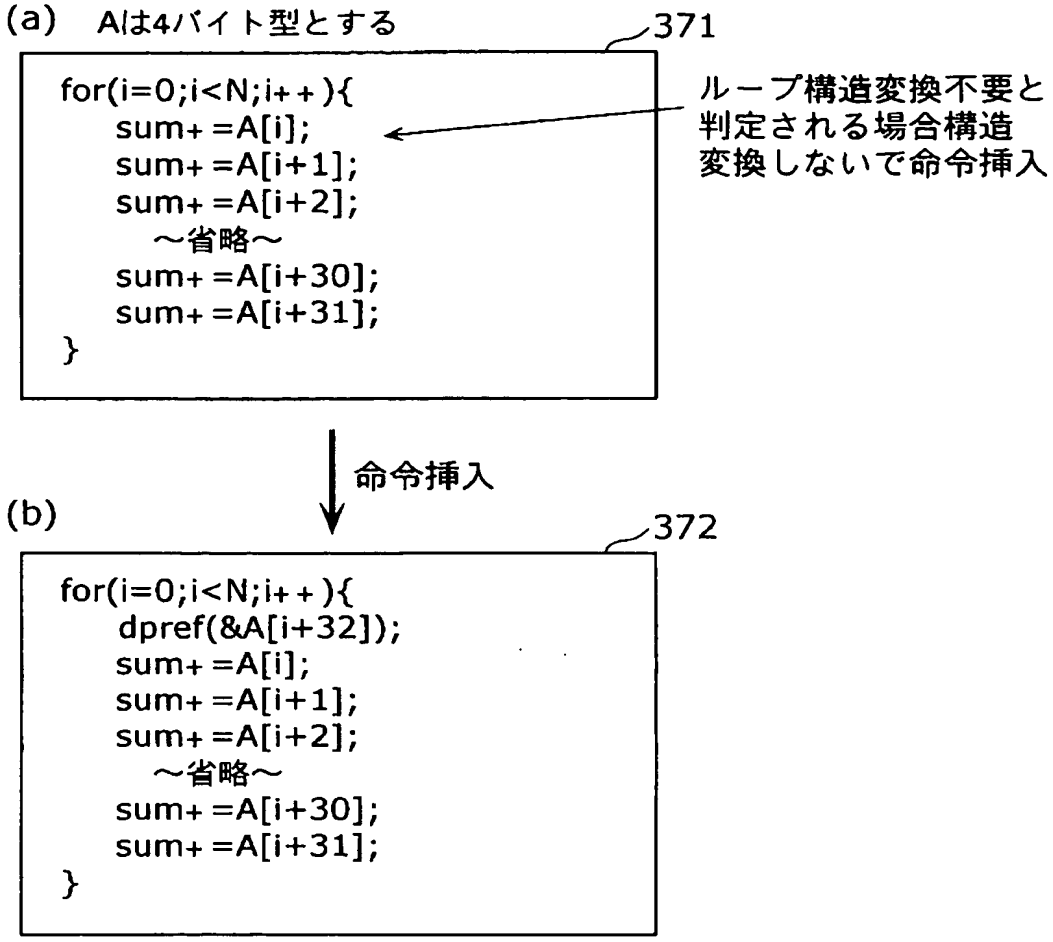
[図22]


(a) Aは4バイト型とする
351
#pragma _min_iteration=128
for(i=0;i<Val;i++){
sum+=A[i];
}
変構換造
(b)
352
for(i=0;i<128;){
for(j=0;j<32;j++,i++){
sum+=A[i];
}
}
for(;i<Val;){
for(j=0;j<32&&i<Val;j++,i++){
sum+=A[i];
}
}
命令挿入
(c)
353
for(i=0;i<128;){
dpref(&A[i+32]);
for(j=0;j<32;j++,i++){
sum+=A[i];
}
}
for(;i<Val){
dpref(&A[i+32]);
for(j=0;j<32&&i<Val;j++,i++){
sum+=A[i];
}
}
ヒーリング置込み
(d)
354
for(i=0;i<Val;){
if(i>=128)N=32;
else N=Val-128;
for(j=0;j<N;j++,i++){
sum+=A[i];
}
}
命令挿入
(e)
355
for(i=0;i<Val;){
dpref(&A[i+32]);
if(i<128)N=32;
else N=Val-128;
for(j=0;j<N;j++,i++){
sum+=A[i];
}
}

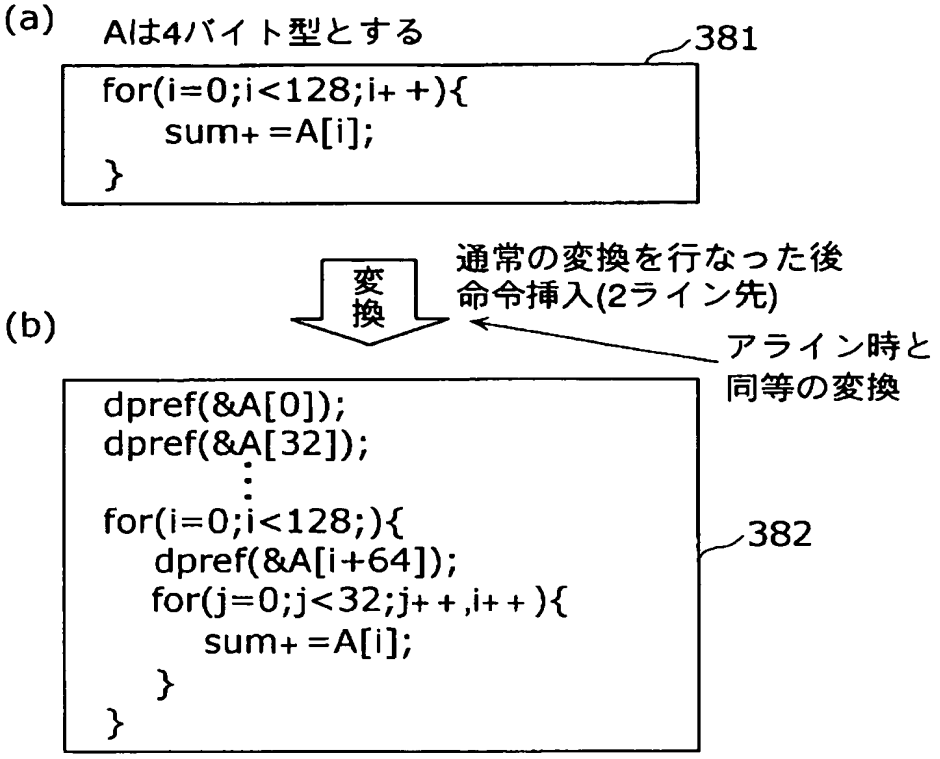
[図23]


(a)
Aは4バイト型とする
361
for(i=0;i<N;i++){
sum+=A[i];
}
変構換造
(b)
362
if(N>1024){
for(i=0;i<1024;){
for(j=0;j<32;j++,i++){
sum+=A[i];
}
}
Val=1024;
}
else{
Val=0;
}
for(i=Val;i<N;i++){
sum+=A[i];
}
動的にループ回数に応じた処理を実行するように変換
命令挿入
(C)
363
if(N>1024){
for(i=0;i<1024;){
dpref(&A[i+32]);
for(j=0;j<32;j++,i++){
sum+=A[i];
}
}
Val=1024;
}
else{
Val=0;
}
for(i=Val;i<N;i++){
sum+=A[i];
}

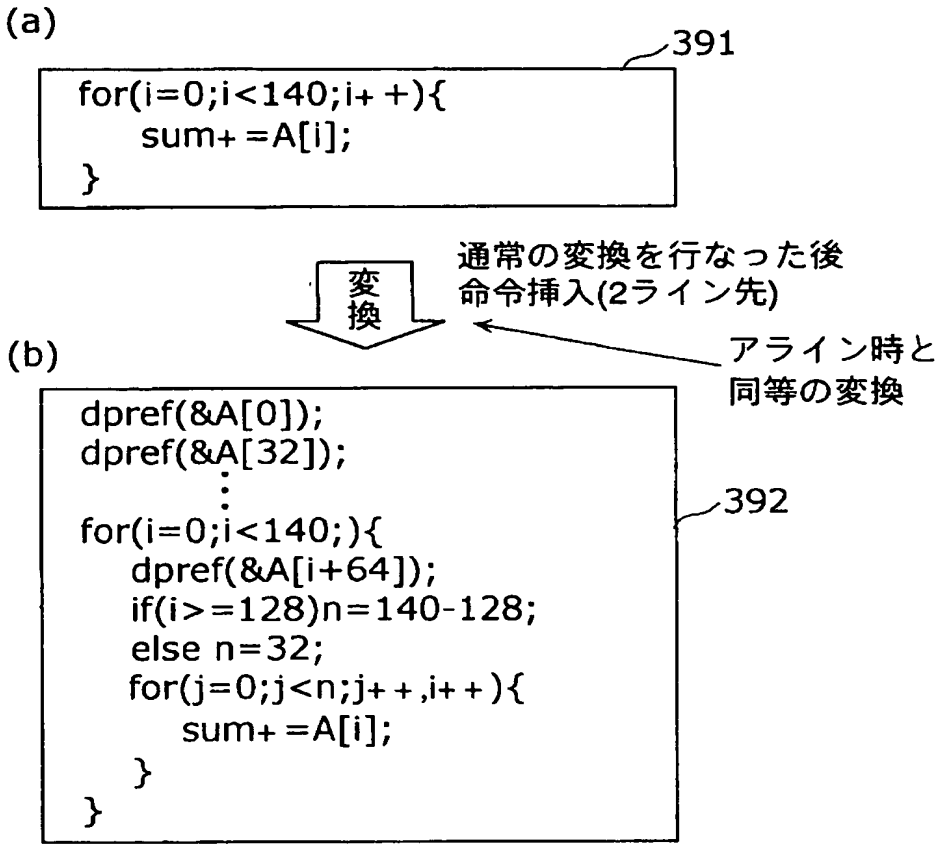
[図24]


(a) Aは4バイト型とする
371
for(i=0;i<N;i++){
sum+=A[i];
sum+=A[i+1];
sum+=A[i+2];
～省略～
sum+=A[i+30];
sum+=A[i+31];
}
ループ構造変換不要と
判定される場合構造
変換しないで命令挿入
命令挿入
(b)
372
for(i=0;i<N;i++){
dpref(&A[i+32]);
sum+=A[i];
sum+=A[i+1];
sum+=A[i+2];
～省略～
sum+=A[i+30];
sum+=A[i+31];
}

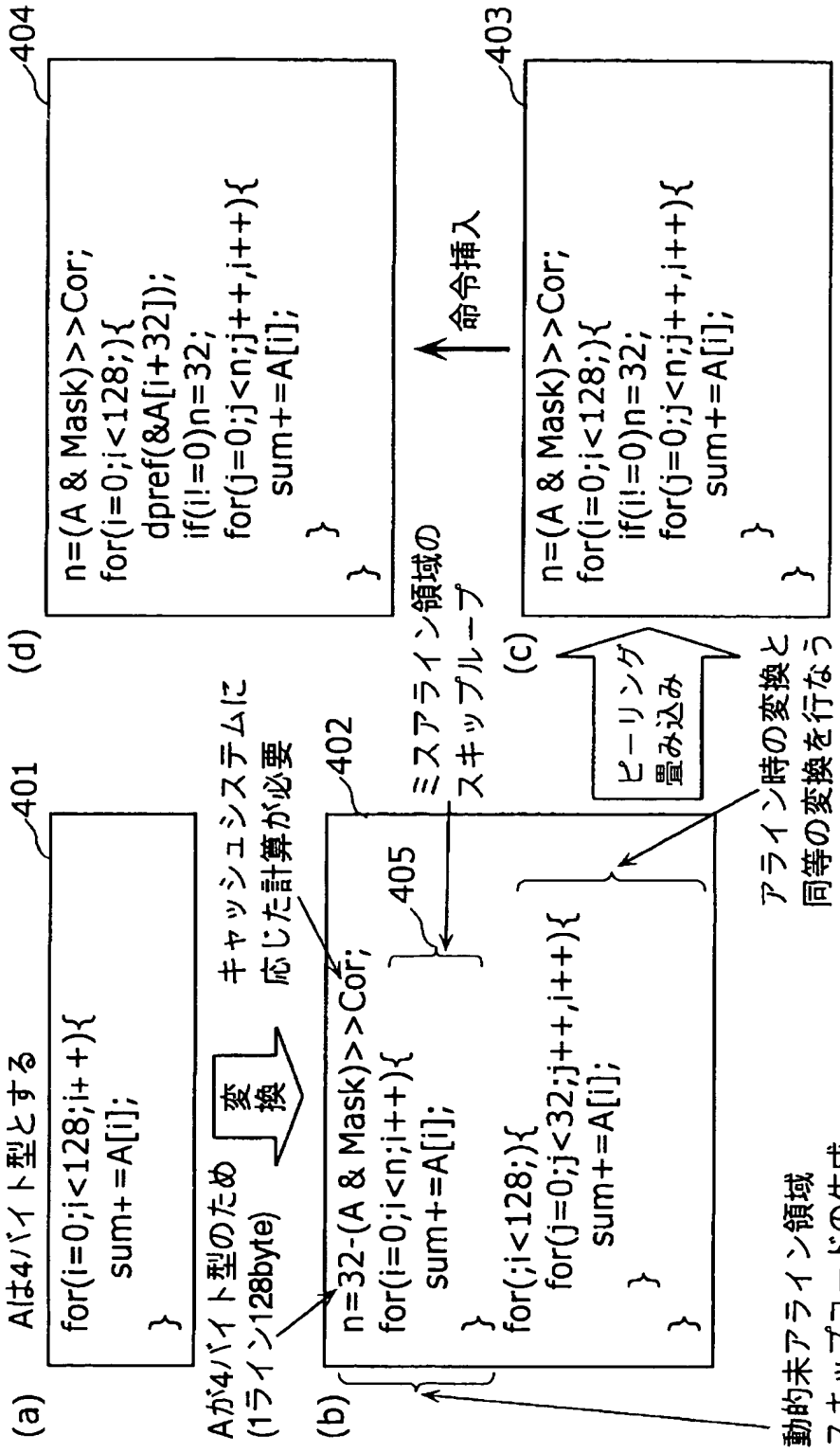
[図25]


(a)
Aは4バイト型とする
381
for(i=0;i<128;i++){
sum+=A[i];
}
変換
通常の変換を行なった後
命令挿入(2ライン先)
アライン時と
同等の変換
(b)
dpref(&A[0]);
dpref(&A[32]);
for(i=0;i<128;){
dpref(&A[i+64]);
for(j=0;j<32;j++,i++){
sum+=A[i];
}
}
382

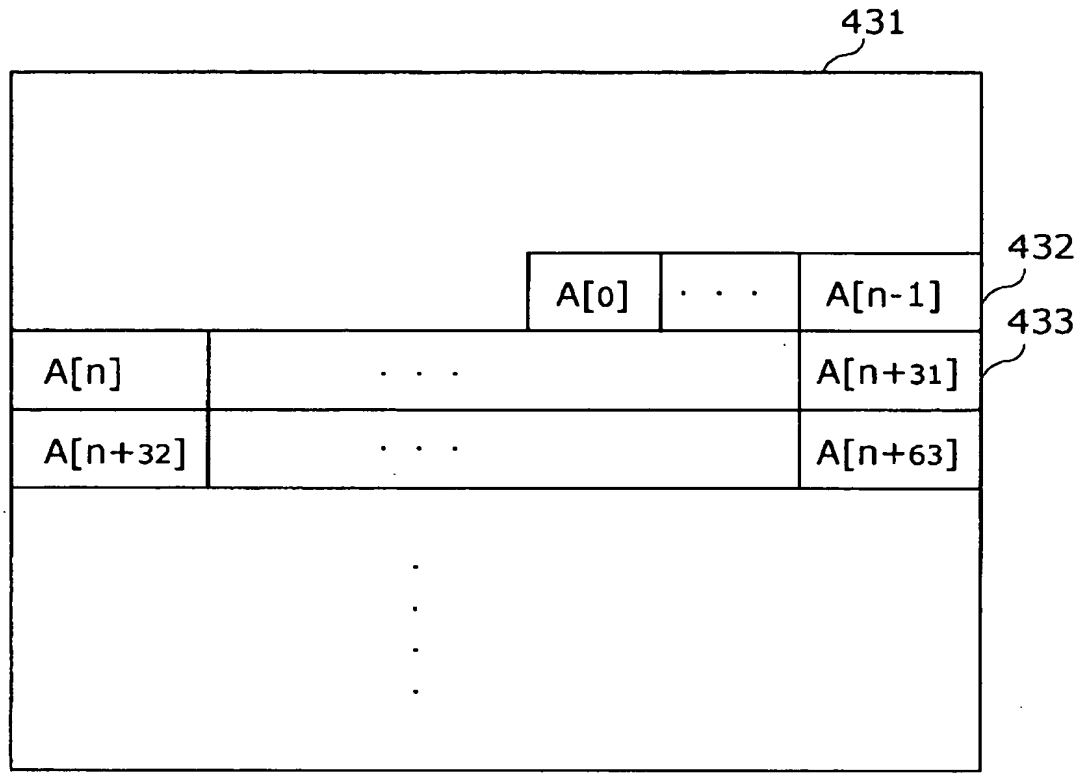
[図26]


(a)
391
for(i=0;i<140;i++){
sum+=A[i];
}
変換
通常の変換を行なった後
命令挿入(2ライン先)
アライン時と
同等の変換
(b)
dpref(&A[0]);
dpref(&A[32]);
:
for(i=0;i<140;){
dpref(&A[i+64]);
if(i>=128)n=140-128;
else n=32;
for(j=0;j<n;j++,i++){
sum+=A[i];
}
}
392

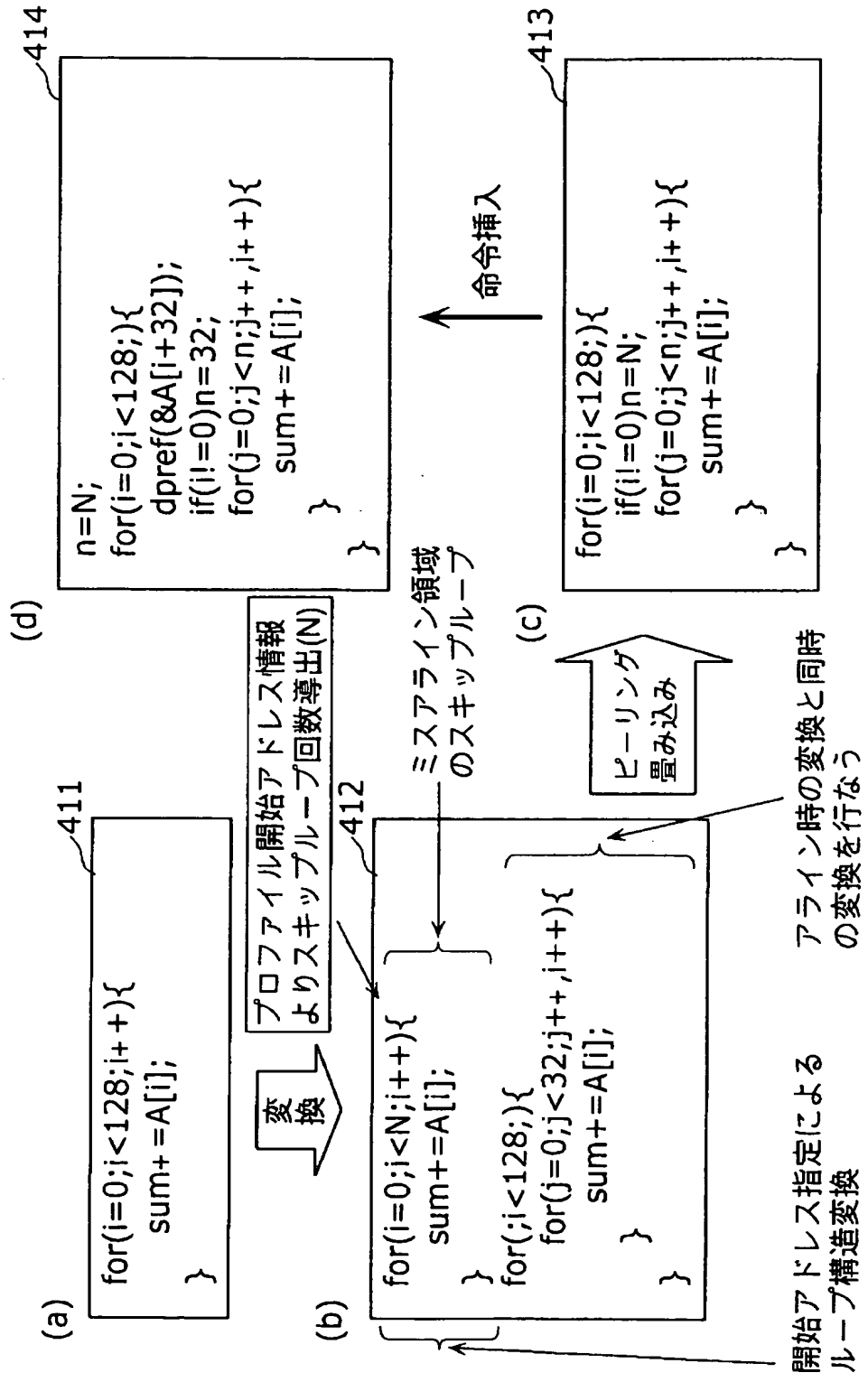
[図27]


(a)
Aは4バイト型とする
401
for(i=0;i<128;i++){
sum+=A[i];
}
Aが4バイト型のため
(1ライン128byte)
変換
キャッシュシステムに
応じた計算が必要
(b)
402
n=32-(A & Mask)>>Cor;
for(i=0;i<n;i++){
sum+=A[i];
}
405
for(;i<128;){
for(j=0;j<32;j++,i++){
sum+=A[i];
}
}
動的末アライン領域
スキップコードの生成
ミスアライン領域の
スキップループ
アライン時の変換と
同等の変換を行なう
(c)
403
ヒーリング
置み込み
n=(A & Mask)>>Cor;
for(i=0;i<128;){
if(i!=0)n=32;
for(j=0;j<n;j++,i++){
sum+=A[i];
}
}
命令挿入
(d)
404
n=(A & Mask)>>Cor;
for(i=0;i<128;){
dpref(&A[i+32]);
if(i!=0)n=32;
for(j=0;j<n;j++,i++){
sum+=A[i];
}
}

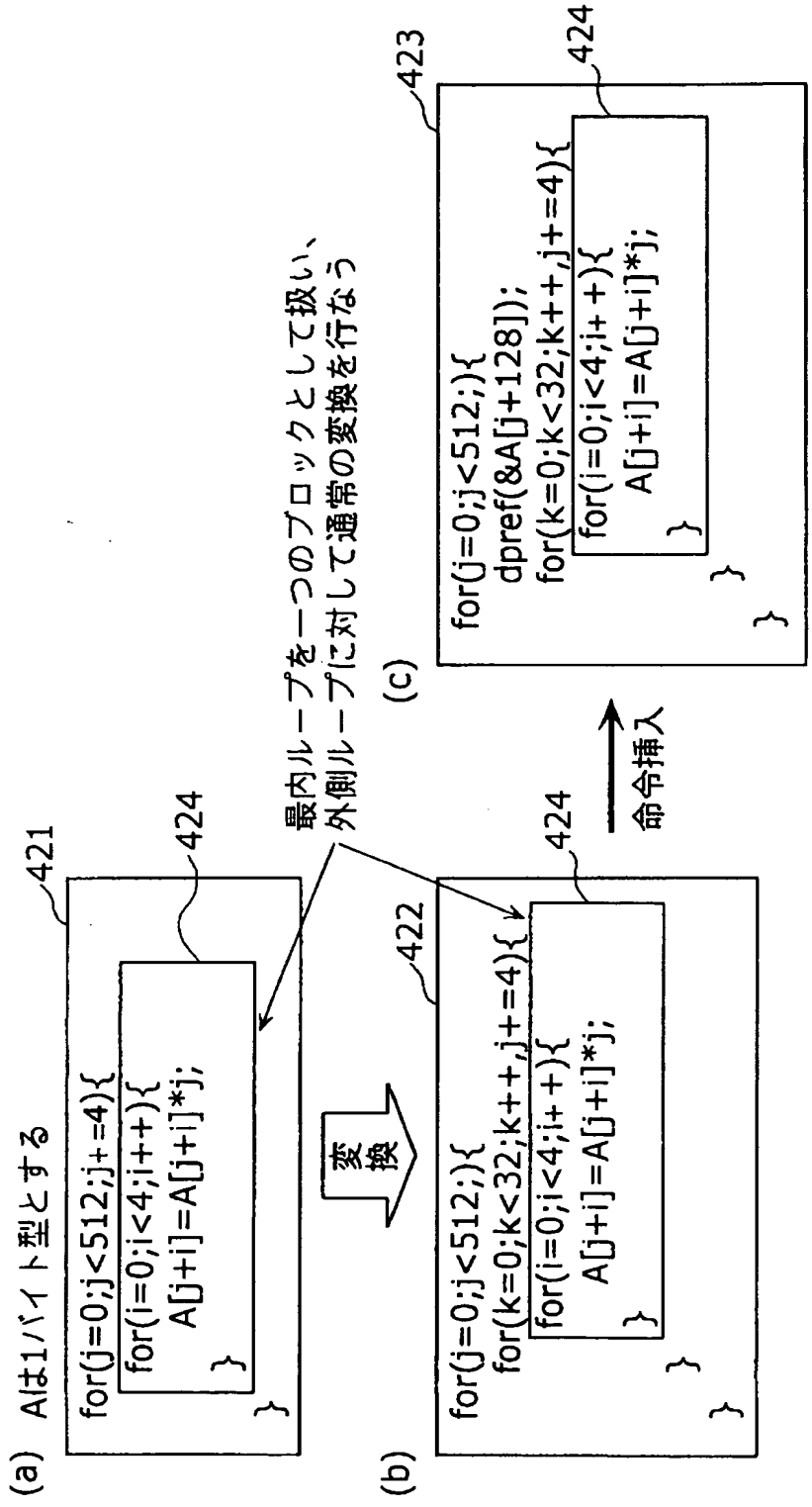
[図28]



[図29]


(a)
411
for(i=0;i<128;i++){
sum+=A[i];
}
変換
プロファイル開始アドレス情報
よりスキップループ回数導出(N)
(b)
412
for(i=0;i<N;i++){
sum+=A[i];
}
for(;i<128;){
for(j=0;j<32;j++,i++){
sum+=A[i];
}
}
ミスアライン領域
のスキップループ
ヒーリング
畳み込み
(c)
413
for(i=0;i<128;){
if(i!=0)n=N;
for(j=0;j<n;j++,i++){
sum+=A[i];
}
}
命令挿入
(d)
414
n=N;
for(i=0;i<128;){
dpref(&A[i+32]);
if(i!=0)n=32;
for(j=0;j<n;j++,i++){
sum+=A[i];
}
}
開始アドレス指定による
ループ構造変換
アライン時の変換と同時
の変換を行なう

[図30]


(a) Aは1バイト型とする
421
424
for(j=0;j<512;j+=4){
for(i=0;i<4;i++){
A[j+i]=A[j+i]*j;
}
}
最内ループを一つのブロックとして扱い、
外側ループに対して通常の変換を行なう
変換
(b)
422
424
for(j=0;j<512;){
for(k=0;k<32;k++,j+=4){
for(i=0;i<4;i++){
A[j+i]=A[j+i]*j;
}
}
}
命令挿入
(c)
423
424
for(j=0;j<512;){
dpref(&A[j+128]);
for(k=0;k<32;k++,j+=4){
for(i=0;i<4;i++){
A[j+i]=A[j+i]*j;
}
}
}

[図31]

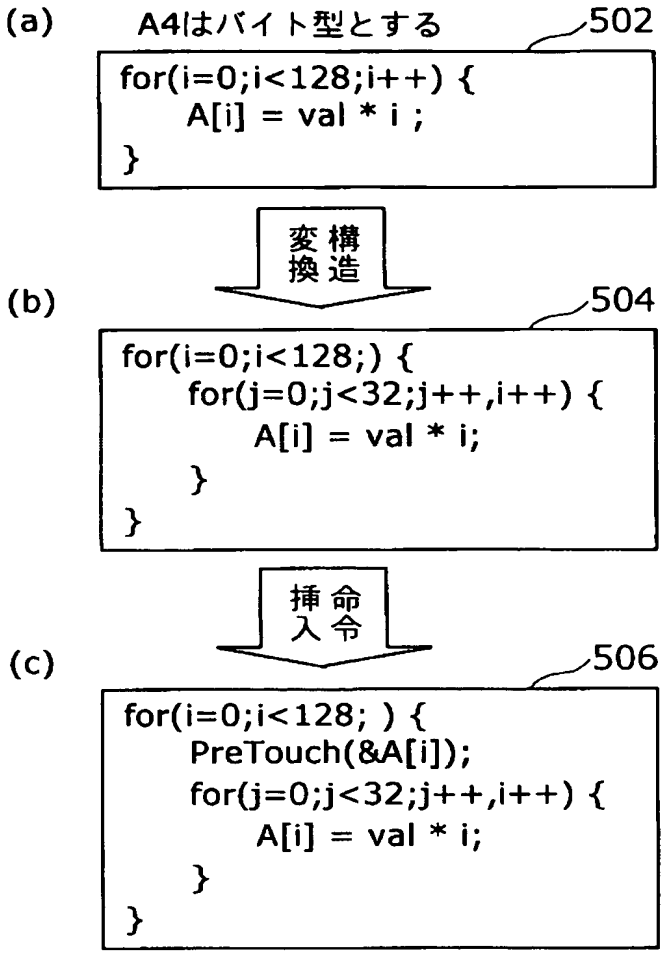
(a)

```
int b[128]:
#pragma _loop_tiling_dpref b
for (i=0; i<128; i++)
{
        a[i] = b[i];
}
```

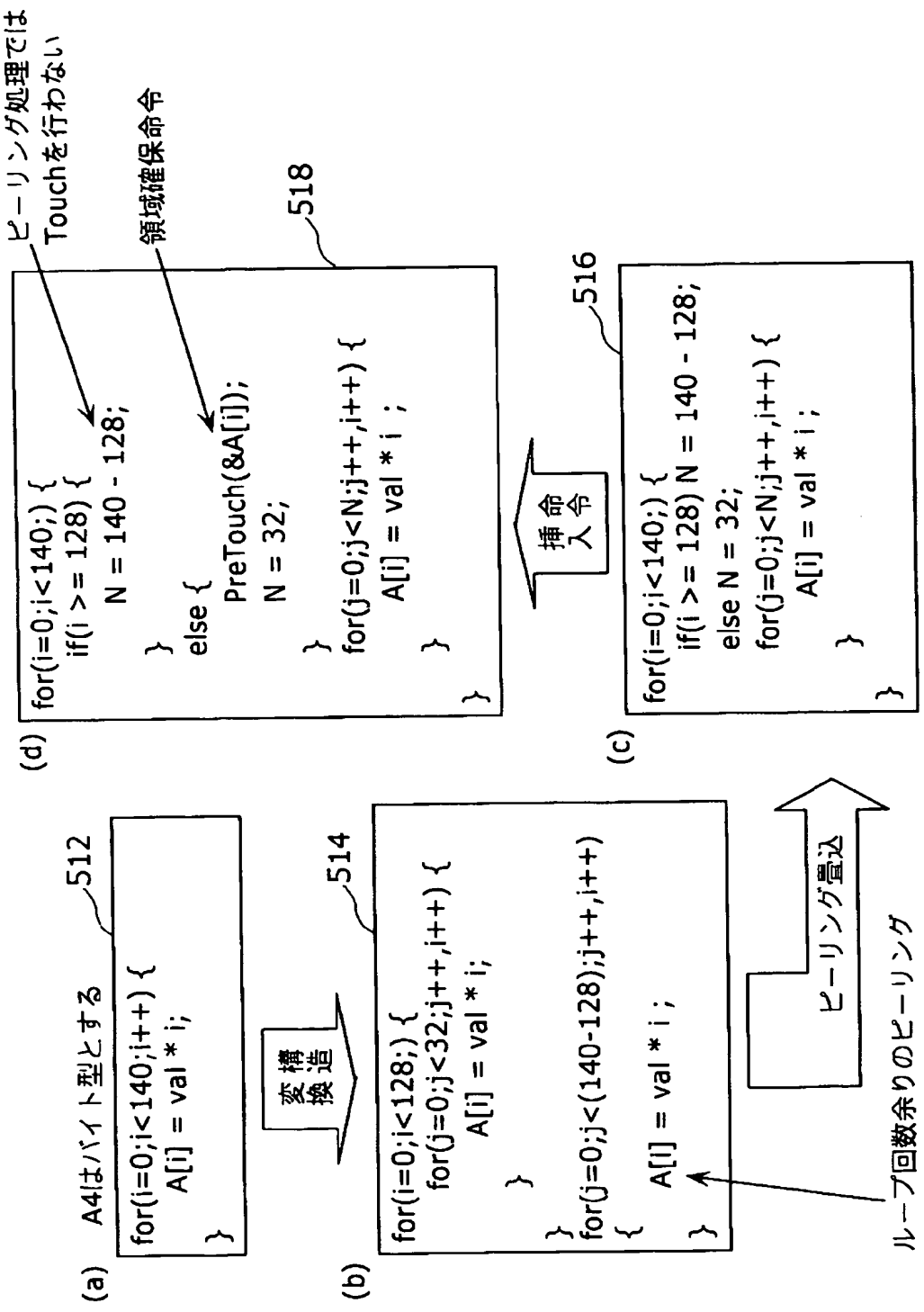
(b)

```
for (i=0; i<128; )
{
        dpref(&b[i+32]);
        for (j =0; j<32; j++, i++){
                a[i] = b[i];
        }
}
```

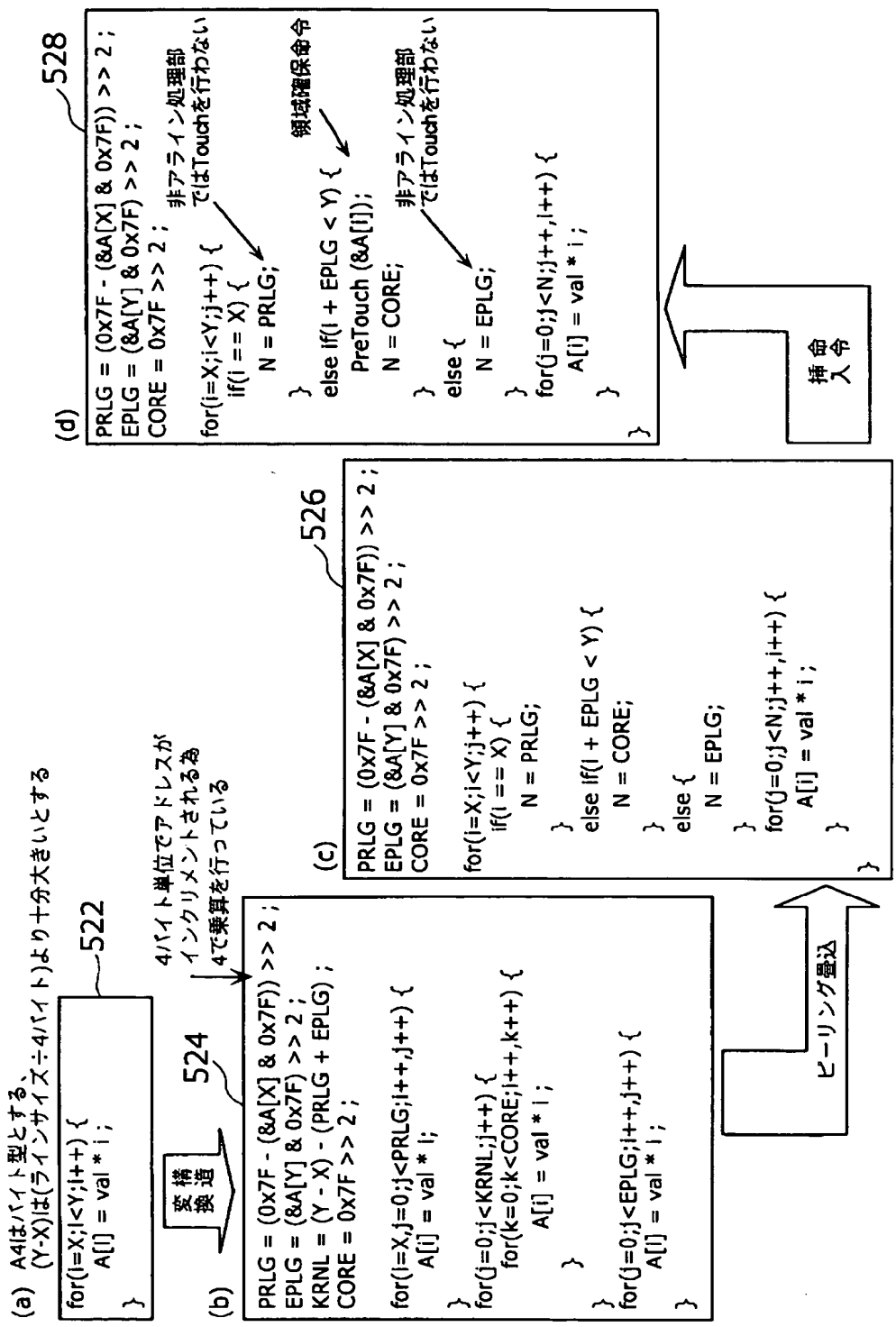
[図32]


(a)
A4はバイト型とする
502
for(i=0;i<128;i++) {
A[i] = val * i ;
}
変構
換造
(b)
504
for(i=0;i<128;) {
for(j=0;j<32;j++,i++) {
A[i] = val * i;
}
}
挿命
入令
(c)
506
for(i=0;i<128;) {
PreTouch(&A[i]);
for(j=0;j<32;j++,i++) {
A[i] = val * i;
}
}

[図33]


(a) A4はバイト型とする
512
for(i=0;i<140;i++) {
A[i] = val * i;
}
変構換造
(b)
514
for(i=0;i<128;) {
for(j=0;j<32;j++,i++) {
A[i] = val * i;
}
}
for(j=0;j<(140-128);j++,i++)
{
A[i] = val * i ;
}
ループ回数余りのヒーリング
ヒーリング置込
(c)
516
for(i=0;i<140;) {
if(i >= 128) N = 140 - 128;
else N = 32;
for(j=0;j<N;j++,i++) {
A[i] = val * i ;
}
}
挿命入令
(d)
518
for(i=0;i<140;) {
if(i >= 128) {
N = 140 - 128;
}
else {
PreTouch(&A[i]);
N = 32;
}
for(j=0;j<N;j++,i++) {
A[i] = val * i ;
}
}
ヒーリング処理ではTouchを行わない
領域確保命令

[図34]


(a) A4はバイト型とする、
(Y-X)は(ラインサイズ÷4バイト)より十分大きいとする
522
for(i=X;i<Y;i++) {
A[i] = val * i ;
}
変換
524
4バイト単位でアドレスが
インクリメントされる為
4で乗算を行っている
(b)
PRLG = (0x7F - (&A[X] & 0x7F)) >> 2 ;
EPLG = (&A[Y] & 0x7F) >> 2 ;
KRNL = (Y - X) - (PRLG + EPLG) ;
CORE = 0x7F >> 2 ;
for(i=X,j=0;j<PRLG;i++,j++) {
A[i] = val * I;
}
for(j=0;j<KRNL;j++) {
for(k=0;k<CORE;i++,k++) {
A[i] = val * I ;
}
}
for(j=0;j<EPLG;i++,j++) {
A[i] = val * I ;
}
ヒーリング量込
526
(c)
PRLG = (0x7F - (&A[X] & 0x7F)) >> 2 ;
EPLG = (&A[Y] & 0x7F) >> 2 ;
CORE = 0x7F >> 2 ;
for(i=X;i<Y;j++) {
if(i == X) {
N = PRLG;
}
else if(I + EPLG < Y) {
N = CORE;
}
else {
N = EPLG;
}
for(j=0;j<N;j++,i++) {
A[i] = val * i ;
}
}
528
(d)
PRLG = (0x7F - (&A[X] & 0x7F)) >> 2 ;
EPLG = (&A[Y] & 0x7F) >> 2 ;
CORE = 0x7F >> 2 ;
for(i=X;i<Y;j++) {
if(i == X) {
N = PRLG;
}
else if(I + EPLG < Y) {
PreTouch (&A[i]);
N = CORE;
}
else {
N = EPLG;
}
for(j=0;j<N;j++,i++) {
A[i] = val * i ;
}
}
非アライン処理部
ではTouchを行わない
領域確保命令
非アライン処理部
ではTouchを行わない
挿 命
入 令